OLmodeling

August 5, 2023

1 Finding a Reliable Model to Predict Offensive Line Play and Draft Position using Feature Engineering and Cross Validation

1.1 Background

In the lead up to the 2023 NFL draft, I was intruigued by the lack of detail in the analysis of Offensive Line prospects. For wide receivers, running backs, and quarterbacks, many high-level analytics were employed to support the argument for the draft position of a player. Such stats include passer rating, yards per touch, and average separation. Due to the inglorious nature of the offensive line positions, bereaved of touchdowns and highlight plays, the evaluation of these players tends to be flippant relative to the aforementioned skill positions.

However, in the NFL, the offensive line is arguably the most important component of a winning formula. Proficent offensive line play seems to be one of the few constant factors when comparing super bowl winners: Chiefs, Rams, Bucs, Patriots. Yes, all these teams had great quarterbacks, but they also had elite offensive line play. The Bucs 2020 line was 5th ranked by PFF with only 24% pressure rate allowed on Tom Brady. The 2021 Rams had the 7th ranked line. Let's see what happened to them once the offensive line was hampered. The Bucs and Rams both had atrocious offenses last season, despite their all-time great quarterbacks. Both teams were bottom 5 in rushing yards last season.

Offensive line play is, in my opinion, the most pivotal factor in determining a contender. Given this, I found it interesting that there's a scarcity in ways to quantify offensive line performance. With skill positions it's easy: receiving yards, rushing yards per carry, etc. I wanted to find a way to predict offensive line play using data from PFF that I had gained access to from being a data collector.

1.2 Research Question

- What features can we choose to best predict the NFL value of an offensive line prospect?

1.3 Data Overview

I created a google sheets dataset from scratch using PFF player data. PFF grades NCAA players on Run Blocking ("RBLK Grade"), Pass Blocking ("PBLK Grade"), and Overall grade("PFF Grade"). I looked at the 2015-2019, 2022 NCAA seasons. For each player, I looked up each of these grades for their college career, and created a data table of prospects. I was faced with many decisions over the course of the data collection: - Which seasons do i use to calculate their overall pass block grade, or run block grade etc. Do I use only the most recent college season, or all of them? - I decided to use all of the seasons available, thinking that the utilization of more data provides a more complete picture of the consistency that a player plays with, making their NFL performance more predictable. One might want to upweight the latest college season to prioritize recent play, but I decided against it. - What amount of plays in a season is needed in order for it to be available to include in my grades? - I decided on 100 plays, thinking that a couple games in a season is enough to judge a player's play for that season.

In addition they track blocking 'Efficiency' ("EFF"): measuring pressure allowed on a per-snap basis with weighting toward sacks allowed. I also entered the Height, Weight, Penalty rate, draft position, college team, and NFL team. These metrics from college performance are all considered potential explanatory variables explaining the players' NFL success.

The response variable, NFL score, was determined by averaging their overall PFF NFL grade with their average PFF Efficiency.

Personal decisions: For each draft year, I upweighted those that played for many seasons, as this is probably an indicator that they're a valuable player. For example, if a player drafted in 2015 has played all of his 8 seasons, I bumped up their NFL score by 5 percent. Similarly, I would downweight a player who only played 2 seasons by a 5 percent decrease. The rules were as follows:

2015 draftees: - 5% increase for more than 5 seasons - 5% decrease for less than 4 seasons

2016 draftees: - 5% increase for more than 4 seasons - 5% decrease for less than 3 seasons

2017 draftees: - 5% increase for more than 3 seasons - 5% decrease for less than 2 seasons

2018 draftees: - 5% increase for more than 3 seasons - 5% decrease for less than 2 seasons

2019 draftees: - 5% increase for more than 3 seasons - 5% decrease for less than 2 seasons

1.4 Data Cleaning and Table Visualization:

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
prospects = pd.read_csv('college_prospect - Sheet1.csv')
```

[3]:
```python
prospects.head()
```

[3]:
```
           Player  PFF Grade  PBLK Grade  RBLK Grade  PEN Rate   EFF Pos  \
0    Ikem Ekwonu      84.30       67.03       90.10      3.30  97.5  OL
1       Evan Neal      80.40       76.23       79.07      2.00  98.6  OL
2  Charles Cross      75.25       77.95       73.75      7.00  97.3  OL
3   Kenyon Green      72.27       56.53       77.20      5.00  97.6  OL
4   Zion Johnson      77.50       74.40       76.70      1.33  98.0  OL

   Drafted  Drafted in Pos College NFL TM  Draft Year  Height  Weight  \
0        6               1    NCST    CAR        2022      76     320
1        7               2     ALA    NYG        2022      79     370
2        9               3  MISSST    SEA        2022      77     305
```

```
3       15                4    TA&M     HOU          2022       76       325
4       17                5      BC     LAC          2022       75       316

   NFL Score     STD
0      80.20   7.677
1      69.35     NaN
2      81.75     NaN
3      66.25     NaN
4      80.95     NaN
```

[4]:
```
STD_NFLscore = 7.677
```

[5]:
```
prospects = prospects.iloc[:,0:15]
```

[6]:
```
prospects.head()
```

[6]:
```
           Player  PFF Grade  PBLK Grade  RBLK Grade  PEN Rate   EFF Pos  \
0     Ikem Ekwonu      84.30       67.03       90.10      3.30  97.5  OL
1       Evan Neal      80.40       76.23       79.07      2.00  98.6  OL
2  Charles Cross      75.25       77.95       73.75      7.00  97.3  OL
3    Kenyon Green      72.27       56.53       77.20      5.00  97.6  OL
4    Zion Johnson      77.50       74.40       76.70      1.33  98.0  OL

   Drafted  Drafted in Pos College NFL TM  Draft Year   Height  Weight  \
0        6               1    NCST    CAR        2022       76     320
1        7               2     ALA    NYG        2022       79     370
2        9               3  MISSST    SEA        2022       77     305
3       15               4    TA&M    HOU        2022       76     325
4       17               5      BC    LAC        2022       75     316

   NFL Score
0      80.20
1      69.35
2      81.75
3      66.25
4      80.95
```

1.5 OLS Regression of NFL score on all possible explanatory variables. (potentially overfit model)

1.5.1 RMSE calculation

[7]:
```
def rmse(actual_y, predicted_y):
    """
    Args:
        predicted_y: an array of the prediction from the model
        actual_y: an array of the groudtruth label
```

```
    Returns:
        The root mean square error between the prediction and the groudtruth
    """
    return np.sqrt(np.mean((actual_y-predicted_y)**2))
```

1.5.2 Split data into training data, test data – fit the model

[8]:
```
from sklearn.model_selection import train_test_split
import sklearn.linear_model as lm
```

[9]:
```
linear_model = lm.LinearRegression()
```

[10]:
```
prospects = prospects.rename({'NFL Score' : 'NFL_score'},axis=1)
```

[11]:
```
y = prospects['NFL_score']
```

[12]:
```
X = prospects.loc[:,['PFF Grade', 'PBLK Grade', 'RBLK Grade', 'EFF','PEN Rate',␣
 ↪'Height', 'Weight']]
X_train1, X_holdout1, Y_train1, Y_holdout1 = train_test_split(X, y, test_size =␣
 ↪0.20)
linear_model.fit(X_train1, Y_train1)
```

[12]:
```
LinearRegression()
```

[13]:
```
train_error1 = rmse(Y_train1, linear_model.predict(X_train1))
holdout_error1 = rmse(Y_holdout1, linear_model.predict(X_holdout1))

print("Training RMSE:", train_error1)
print("Holdout RMSE:", holdout_error1)
```

```
Training RMSE: 9.018430867852945
Holdout RMSE: 9.120086963880954
```

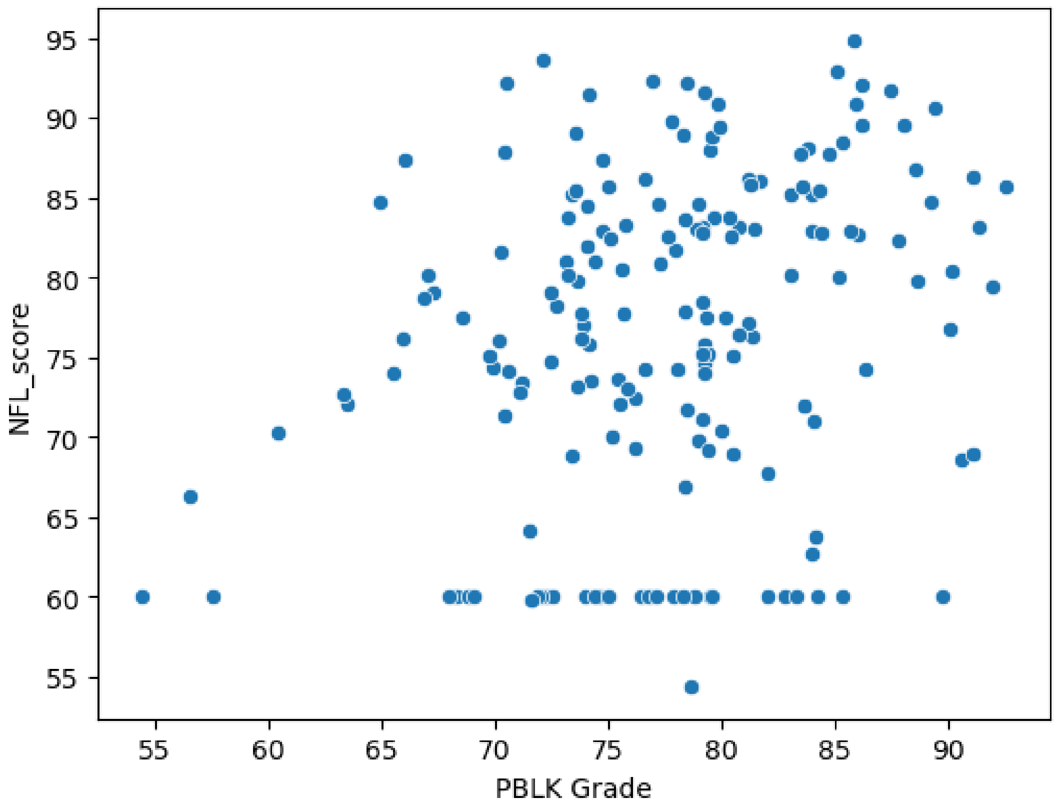
Here the train RMSE and Holdout RMSE indicates that the preliminary model is off by an average of 9.12 points when predicting NFL score using all EFF, PBLK, RBLK, PEN Rate, Height, and Weight as predictive features

1.5.3 EDA : Examine pairwise correlations between NFL score and possible features in order to eliminate features from the overfit model that are not of use

Scatterplots:

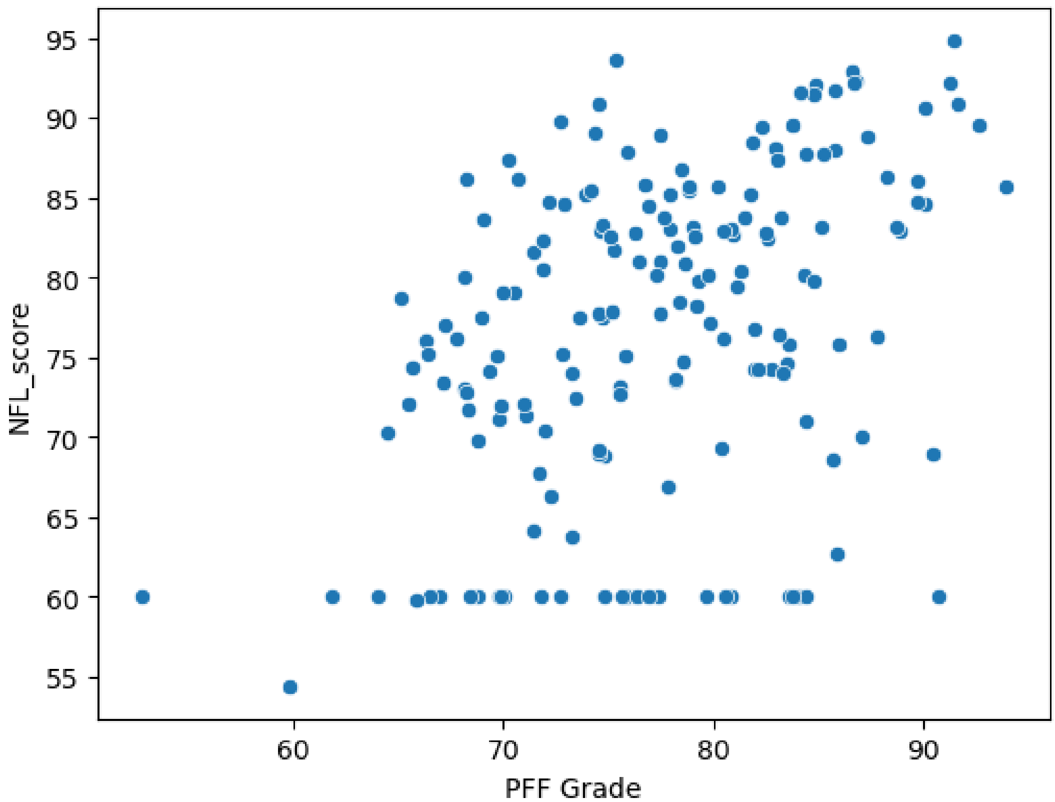
[14]:
```
sns.scatterplot(data = prospects, x = 'PBLK Grade', y = 'NFL_score')
```

[14]:
```
<AxesSubplot:xlabel='PBLK Grade', ylabel='NFL_score'>
```



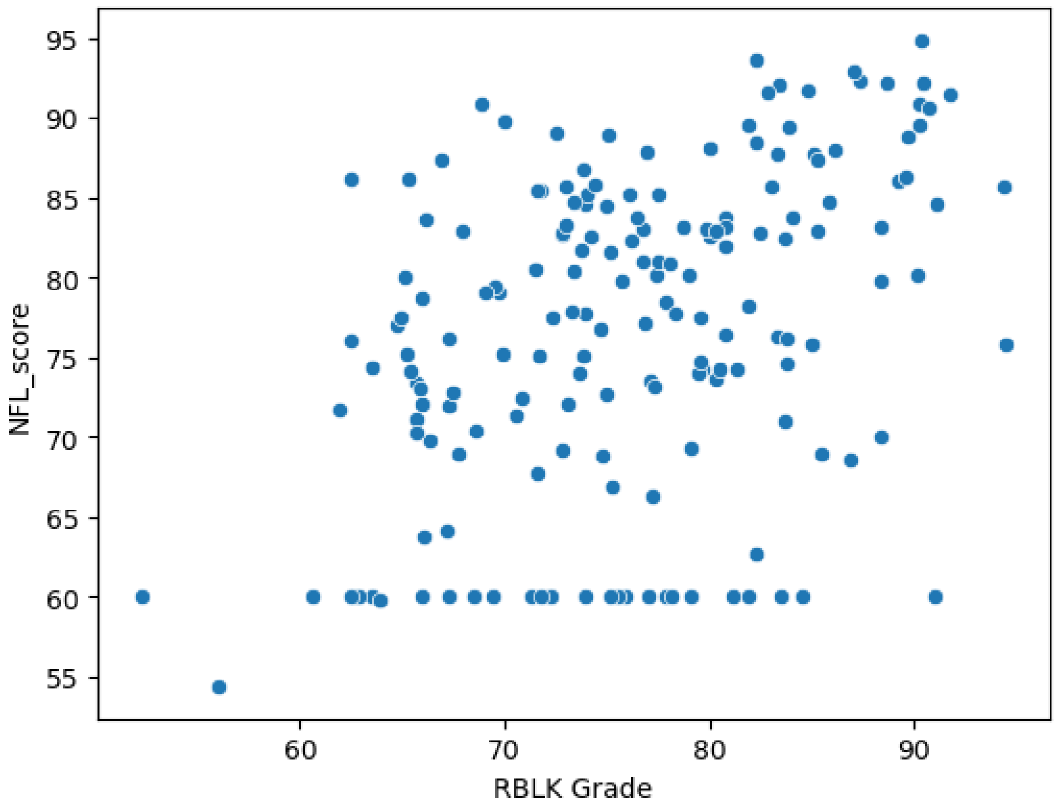
[15]:
```
sns.scatterplot(data = prospects, x = 'PFF Grade', y = 'NFL_score')
```

[15]: <AxesSubplot:xlabel='PFF Grade', ylabel='NFL_score'>



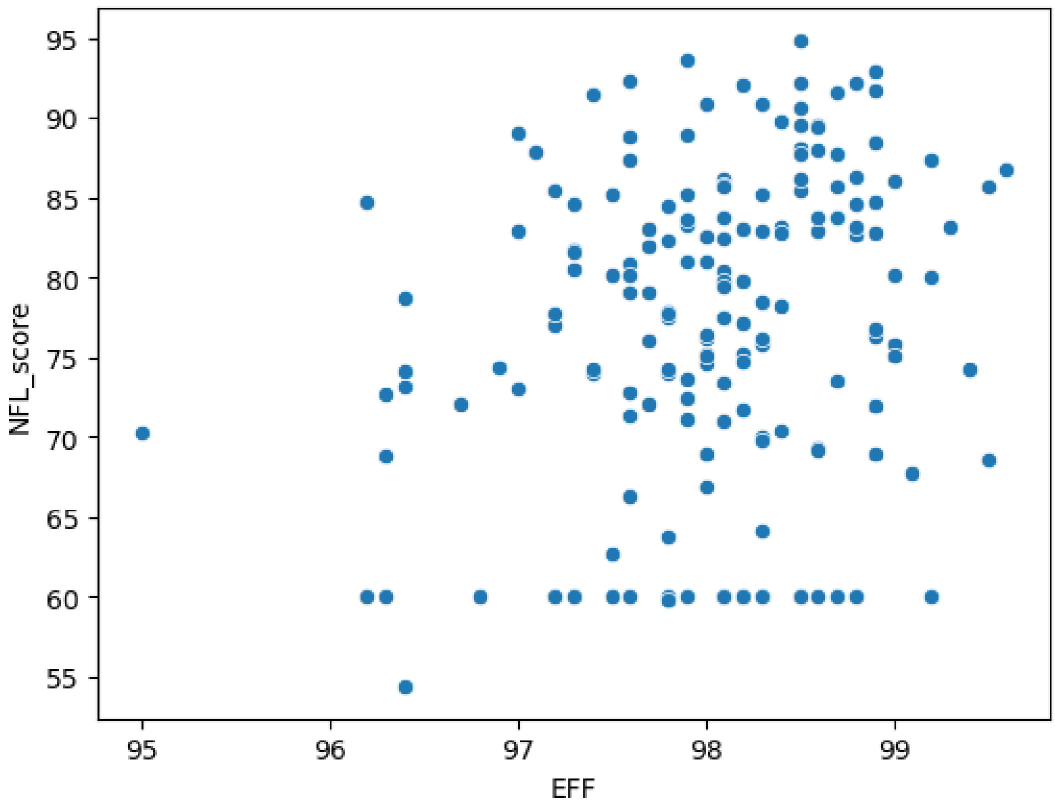
[16]:
```
sns.scatterplot(data = prospects, x = 'RBLK Grade', y = 'NFL_score')
```

[16]:
```
<AxesSubplot:xlabel='RBLK Grade', ylabel='NFL_score'>
```



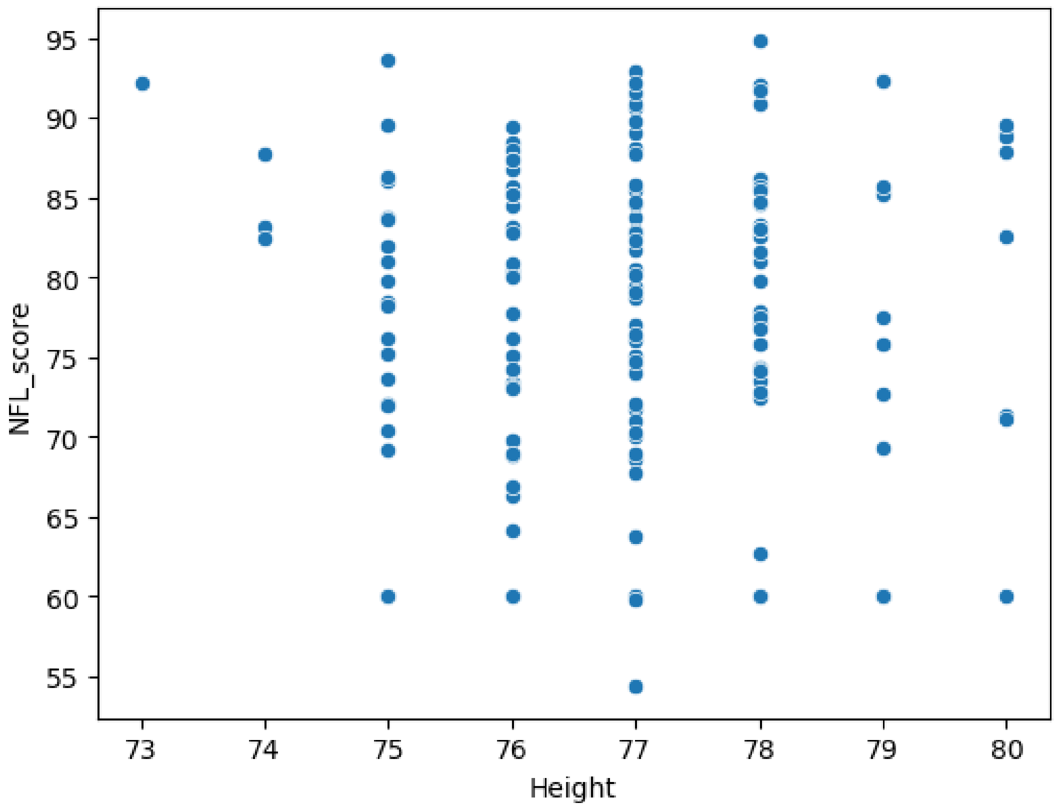
[17]:
```
sns.scatterplot(data = prospects, x = 'EFF', y = 'NFL_score')
```

[17]: <AxesSubplot:xlabel='EFF', ylabel='NFL_score'>



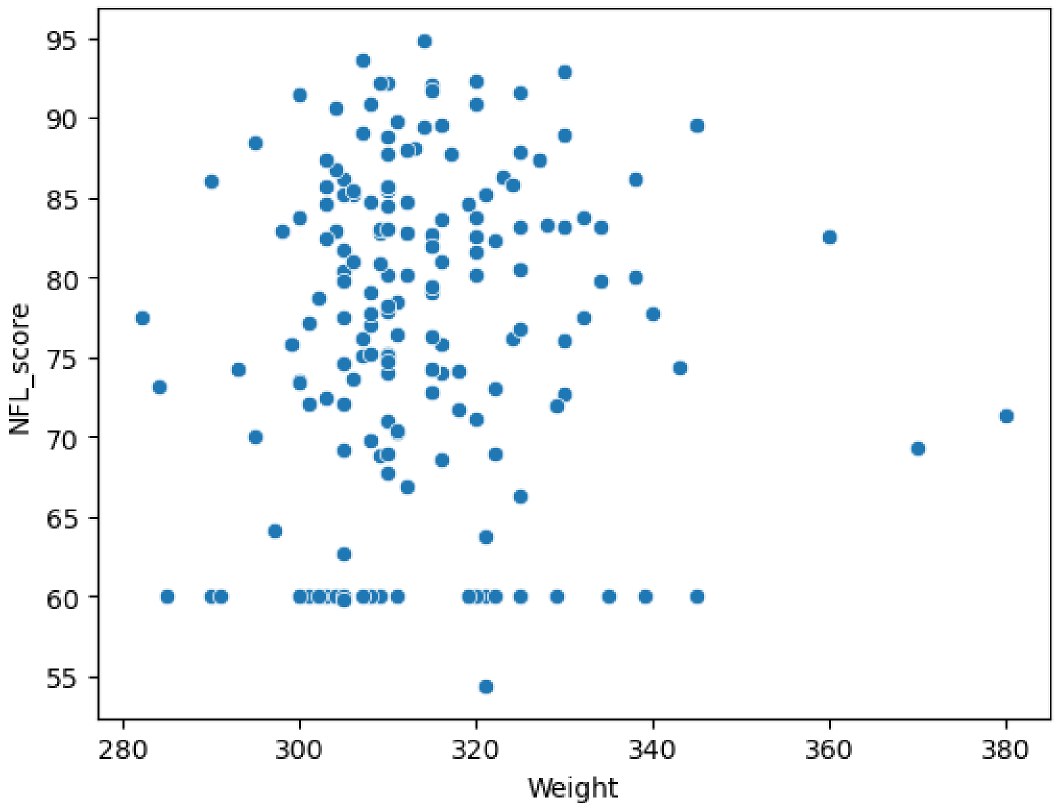

```
[18]: sns.scatterplot(data = prospects, x = 'Height', y = 'NFL_score')
```

```
[18]: <AxesSubplot:xlabel='Height', ylabel='NFL_score'>
```



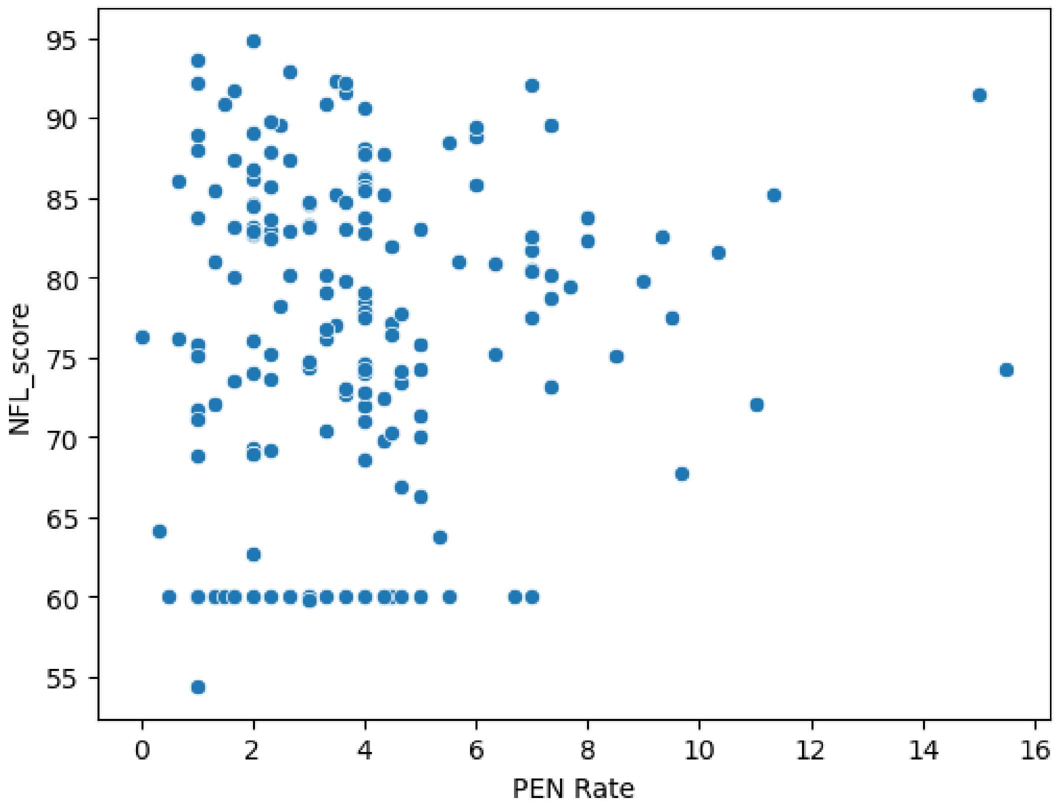
[19]:
```
sns.scatterplot(data = prospects, x = 'Weight', y = 'NFL_score')
```

[19]: <AxesSubplot:xlabel='Weight', ylabel='NFL_score'>



[20]:
```
sns.scatterplot(data = prospects, x = 'PEN Rate', y = 'NFL_score')
```

[20]: <AxesSubplot:xlabel='PEN Rate', ylabel='NFL_score'>



Correlations given by the pearson coefficient:

[21]:
```
np.corrcoef(prospects['PBLK Grade'], prospects['NFL_score'])[0,1]
```

[21]: 0.26383126061026685

[22]:
```
np.corrcoef(prospects['PFF Grade'], prospects['NFL_score'])[0,1]
```

[22]: 0.41323335401210937

[23]:
```
np.corrcoef(prospects['RBLK Grade'], prospects['NFL_score'])[0,1]
```

[23]: 0.41797492291931826

[24]:
```
np.corrcoef(prospects['EFF'], prospects['NFL_score'])[0,1]
```

[24]: 0.21331969632505463

The scatterplots indicate that Height, Weight, and PEN Rate have approximately zero correlation with NFL score. This indicates that they are useless features in our regression analysis, since for example an increase in Height gives no extra information about the change in NFL score

Conversely, the plots and Pearson correlation coeefficients reveal that PFF Grade and RBLK Grade have moderately strong positive correlations with NFL score, indicating its efficacy as a potential regression feature. Also, EFF and PBLK Grade have a weak positive correlation with NFL score, but could be useful in contributing to the OLS regression estimated NFL scores in tandem with either RBLK Grade or PFF Grade.

NEW POTENTIAL FEATURES AFTER EDA: PBLK Grade, RBLK Grade, PFF Grade, EFF

We will now explore the training error and Holdout error on every combination of these potential features to deduce which one gives us the the best holdout error.

We desire a model that gives a low holdout error for the following reasons: - Training error too small indicates overfitting with a lack of generalizability of predictions to new samples of players, resulting in a larger holdout error - Training error too large indicates a poor fit of the regression line to the training data, and thus a failure to learn patterns from the given data

1.6 Training the Models

[27]:
```
errors_vs_whichFeatures = pd.DataFrame(columns = ["Features", "Training Error",␣
↪"Holdout Error"])

def train_mod_w_these_features(features):
    """
    Args:
        features: The features used in this regression model to predict NFL play


    Returns:
        The training error and holdout error given by fitting a regression␣
↪model with these select features
    """
    X_train_chosen_features = X_train1.loc[:, features]

    linear_model.fit(X_train_chosen_features, Y_train1)
    train_error = rmse(Y_train1, linear_model.predict(X_train_chosen_features))

    X_holdout_chosen_features = X_holdout1.loc[:, features]
    holdout_error = rmse(Y_holdout1, linear_model.
↪predict(X_holdout_chosen_features))
    errors_vs_whichFeatures.loc[len(errors_vs_whichFeatures)] = [features,␣
↪train_error, holdout_error]
```

[28]:
```
train_mod_w_these_features(['PBLK Grade'])
train_mod_w_these_features(['PBLK Grade', 'RBLK Grade'])
train_mod_w_these_features(['PBLK Grade', 'RBLK Grade', 'PFF Grade'])
train_mod_w_these_features(['PBLK Grade', 'RBLK Grade', 'PFF Grade', 'EFF'])
train_mod_w_these_features(['PBLK Grade', 'PFF Grade'])
```

```
train_mod_w_these_features(['PBLK Grade', 'EFF'])
train_mod_w_these_features(['RBLK Grade'])
train_mod_w_these_features(['RBLK Grade', 'PFF Grade'])
train_mod_w_these_features(['RBLK Grade', 'EFF'])
train_mod_w_these_features(['PFF Grade'])
train_mod_w_these_features(['PFF Grade', 'EFF'])
train_mod_w_these_features(['PBLK Grade', 'RBLK Grade', 'EFF'])
train_mod_w_these_features(['PBLK Grade', 'EFF', 'PFF Grade'])
train_mod_w_these_features(['EFF', 'RBLK Grade', 'PFF Grade'])
train_mod_w_these_features(['EFF'])
```

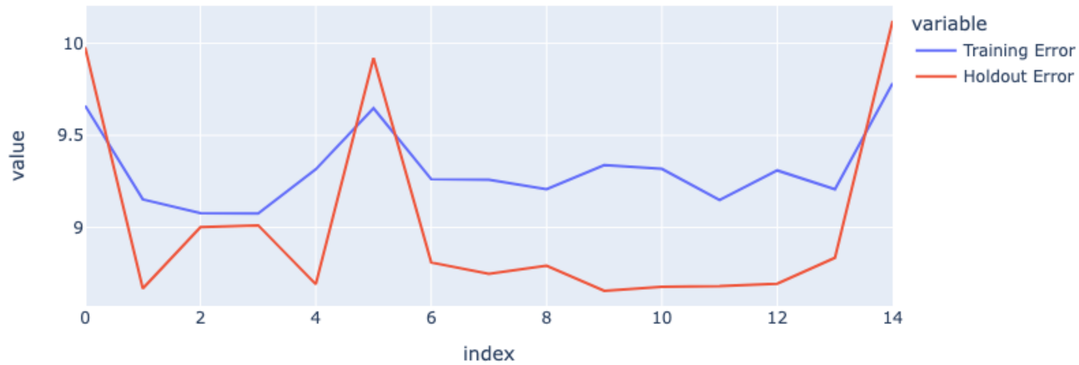
[29]:
```
errors_vs_whichFeatures
```

[29]:

	Features	Training Error	Holdout Error
0	[PBLK Grade]	9.661503	9.977008
1	[PBLK Grade, RBLK Grade]	9.152810	8.669047
2	[PBLK Grade, RBLK Grade, PFF Grade]	9.078472	9.001977
3	[PBLK Grade, RBLK Grade, PFF Grade, EFF]	9.076255	9.011761
4	[PBLK Grade, PFF Grade]	9.316016	8.691987
5	[PBLK Grade, EFF]	9.646930	9.919388
6	[RBLK Grade]	9.262195	8.810240
7	[RBLK Grade, PFF Grade]	9.259152	8.748074
8	[RBLK Grade, EFF]	9.208276	8.792037
9	[PFF Grade]	9.338697	8.655614
10	[PFF Grade, EFF]	9.317809	8.677987
11	[PBLK Grade, RBLK Grade, EFF]	9.149361	8.681059
12	[PBLK Grade, EFF, PFF Grade]	9.309550	8.694613
13	[EFF, RBLK Grade, PFF Grade]	9.206992	8.835616
14	[EFF]	9.783951	10.121870

[30]:
```
top_4 = [['PBLK Grade', 'RBLK Grade', 'PFF Grade'],['PBLK Grade', 'RBLK Grade',␣
↪'PFF Grade', 'EFF'],['PBLK Grade', 'RBLK Grade'],['PBLK Grade', 'RBLK␣
↪Grade', 'EFF']]
```

[31]:
```
import plotly.express as px
px.line(errors_vs_whichFeatures, x = errors_vs_whichFeatures.index, y =␣
↪["Training Error", "Holdout Error"])
```



The best models appear to have a holdout error below 8.8, which is how we'll choose to delineate our potential models from our eliminated models moving forward

[34]:
```
candidates_best_mod = errors_vs_whichFeatures[errors_vs_whichFeatures['Holdout␣
↪Error'] <8.8]
candidates_best_mod
```

[34]:

	Features	Training Error	Holdout Error
1	[PBLK Grade, RBLK Grade]	9.152810	8.669047
4	[PBLK Grade, PFF Grade]	9.316016	8.691987
7	[RBLK Grade, PFF Grade]	9.259152	8.748074
8	[RBLK Grade, EFF]	9.208276	8.792037
9	[PFF Grade]	9.338697	8.655614
10	[PFF Grade, EFF]	9.317809	8.677987
11	[PBLK Grade, RBLK Grade, EFF]	9.149361	8.681059
12	[PBLK Grade, EFF, PFF Grade]	9.309550	8.694613

The models that use these features above are the remaining candidates. The model that uses PFF Grade and PBLK as predictors, and the model that uses PBLK Grade and RBLK Grade as predictors are the best candidates so far (lowest validation error with a reasonably low training error)

[35]:
```
candidates_best_mod['train_hold_ratio'] = candidates_best_mod['Holdout Error']/
↪candidates_best_mod['Training Error']
```

```
/tmp/ipykernel_96/2265175316.py:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[36]:
```
candidates_best_mod
```

[36]:
```
                           Features  Training Error  Holdout Error  \
1         [PBLK Grade, RBLK Grade]        9.152810       8.669047
4          [PBLK Grade, PFF Grade]        9.316016       8.691987
7          [RBLK Grade, PFF Grade]        9.259152       8.748074
8                [RBLK Grade, EFF]        9.208276       8.792037
9                      [PFF Grade]        9.338697       8.655614
10                [PFF Grade, EFF]        9.317809       8.677987
11   [PBLK Grade, RBLK Grade, EFF]        9.149361       8.681059
12    [PBLK Grade, EFF, PFF Grade]        9.309550       8.694613

    train_hold_ratio
1           0.947146
4           0.933015
7           0.944803
8           0.954797
9           0.926855
10          0.931333
11          0.948816
12          0.933946
```

1.7 Regularization

Among the model candidates above, there are a variety of options – from models that use up to 4 features to models that use only one feature. In general, if there are two models that perform similarly but with different numbers of features, it's more intuitive to use the model with less features because it is less likely to be shown to be overfit with more test sets.

So, I proceeded to regularize the OLS regression model, using Ridge regression on the full repetoire of features. This is to say that the sum of the parameters of the model was given a ceiling. The constricting of the parameters prevents overfitting.

I regularized the model at different levels of extremity (by altering the 'alpha' value), and examined how the model performed (holdout error) acordingly. This intented to identify approximately which level of regularization is conducive to best performance, giving more insight into the number of features that is optimal to choose when examiming our model candidates again.

first, we must adjust all the parameters to the same scale in order to carry out Ridge Regression

[37]:
```
from sklearn.preprocessing import StandardScaler
prosp_forscale= prospects.loc[:,['PFF Grade', 'PBLK Grade', 'RBLK Grade',
 ↪'EFF','PEN Rate', 'Height', 'Weight']]
ss = StandardScaler()
```

```
ss.fit(prosp_forscale)
prospects_scaled = pd.DataFrame(ss.transform(prosp_forscale), columns =␣
 ↪prosp_forscale.columns)
prospects_scaled
```

[37]:

```
     PFF Grade  PBLK Grade  RBLK Grade       EFF  PEN Rate    Height    Weight
0     0.939845   -1.524476    1.689066 -0.729592 -0.227801 -0.681492  0.461987
1     0.410103   -0.211389    0.354883  0.768842 -0.743341  1.511674  4.128882
2    -0.289428    0.034101   -0.288622 -1.002034  1.239506  0.049563 -0.638081
3    -0.694205   -3.023109    0.128689 -0.593371  0.446367 -0.681492  0.828677
4     0.016192   -0.472579    0.068209 -0.048486 -1.009043 -1.412548  0.168636
..         …           …           …         …         …         …         …
172  -1.242963   -0.939296   -1.048247 -0.593371  0.049797  0.780619  0.095298
173  -1.016125   -0.386943   -1.068810  0.632621 -0.874209 -0.681492 -0.491405
174  -0.713222   -1.818494   -0.330957 -2.500468 -0.081070  0.780619 -0.418067
175  -0.387227    0.251046   -0.399904  0.768842 -0.612474 -1.412548 -0.638081
176  -1.097624   -1.010660   -1.295005 -2.228025  0.315499  0.780619  0.315312

[177 rows x 7 columns]
```

now train the models at the varying levels of alpha (greater numbers of alpha corresponds to more regularization of the parameters)

[38]:

```
X = prospects_scaled
X_train2, X_holdout2, Y_train2, Y_holdout2 = train_test_split(X, y, test_size =␣
 ↪0.20)
```

[39]:

```
from sklearn.linear_model import Ridge
error_vs_alpha = pd.DataFrame(columns = ["alpha", "Training Error", "Holdout␣
 ↪Error"])
range_of_alphas = 10**np.linspace(-5, 4, 40)

for alpha in range_of_alphas:
    linear_model_reg = Ridge(alpha)
    linear_model_reg.fit(X_train2,Y_train2)
    pred_train = linear_model_reg.predict(X_train2)
    pred_holdout = linear_model_reg.predict(X_holdout2)
    err_train = rmse(Y_train2, pred_train)
    err_holdout = rmse(Y_holdout2, pred_holdout)
    error_vs_alpha.loc[len(error_vs_alpha)] = [alpha, err_train, err_holdout]

error_vs_alpha.head()
```

[39]:

```
      alpha  Training Error  Holdout Error
0  0.000010        8.987461       9.406277
1  0.000017        8.987461       9.406275
2  0.000029        8.987461       9.406273
```

```
3  0.000049        8.987461       9.406269
4  0.000084        8.987461       9.406263
```

[40]:
```
error_vs_alpha[20:30]
```

[40]:

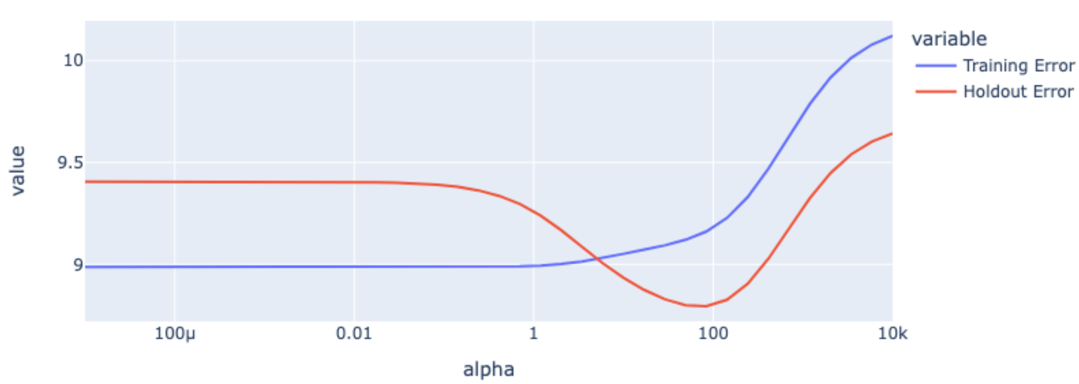
	alpha	Training Error	Holdout Error
20	0.412463	8.988581	9.336191
21	0.701704	8.990305	9.295780
22	1.193777	8.994173	9.239171
23	2.030918	9.001781	9.167224
24	3.455107	9.014421	9.086139
25	5.878016	9.031780	9.005653
26	10.000000	9.051713	8.933968
27	17.012543	9.072144	8.874621
28	28.942661	9.093588	8.828607
29	49.238826	9.120497	8.799106

[41]:
```
error_vs_alpha[30:40]
```

[41]:

	alpha	Training Error	Holdout Error
30	83.767764	9.161379	8.794376
31	142.510267	9.227734	8.826341
32	242.446202	9.330095	8.904597
33	412.462638	9.469540	9.027481
34	701.703829	9.630617	9.176680
35	1193.776642	9.787101	9.324319
36	2030.917621	9.917483	9.448223
37	3455.107295	10.013879	9.540110
38	5878.016072	10.079373	9.602629
39	10000.000000	10.121457	9.642829

[42]:
```
px.line(error_vs_alpha, x = "alpha", y = ["Training Error", "Holdout Error"],
 ↪log_x=True)
```



it appears that an alpha value of around 83 is optimal for modeling, as it corresponds with a decrease in Holdout error and only sacrificing a minimal increase in training error.

So, this indicates that we want a moderate amount of regularization. Not too much too the point of accelerating the training error, but not too little to the point where we cannot get the decrease in holdout error that we desire. This gives us an inclination that the model that uses [RBLK Grade, PFF Grade] as its features to predict NFL score would be an optimal choice.

1.8 Cross-Validation Error

Now that we have an idea of the number parameters that we want to use, and which combinations of features are candidates for the best model, we can also use Cross Validation to evaluate some of these candidate models.

For a specified model, this Cross Validation splits the training set into 4 folds, and assigns the holdout set to be one of these 4 parts on each pass through. The holdout error is given by the average of the holdout errors given on each iteration. We will use these holdout errors to asses our candidate models with their specified features.

[43]:
```
from sklearn.model_selection import KFold

def compute_CV_error(model, X_train, Y_train):
    '''
    Split the training data into 4 subsets.
    For each subset,
        fit a model holding out that subset
        compute the MSE on that subset (the validation set)
    You should be fitting 4 models total.
    Return the average MSE of these 4 folds.

    Args:
        model: an sklearn model with fit and predict functions
        X_train (data_frame): Training data
        Y_train (data_frame): Label

    Return:
        the average validation MSE for the 4 splits.
    '''
    kf = KFold(n_splits=4)
    validation_errors = []

    for train_idx, valid_idx in kf.split(X_train):
        # split the data
        split_X_train, split_X_valid = X_train.iloc[train_idx], X_train.
↪iloc[valid_idx]
```

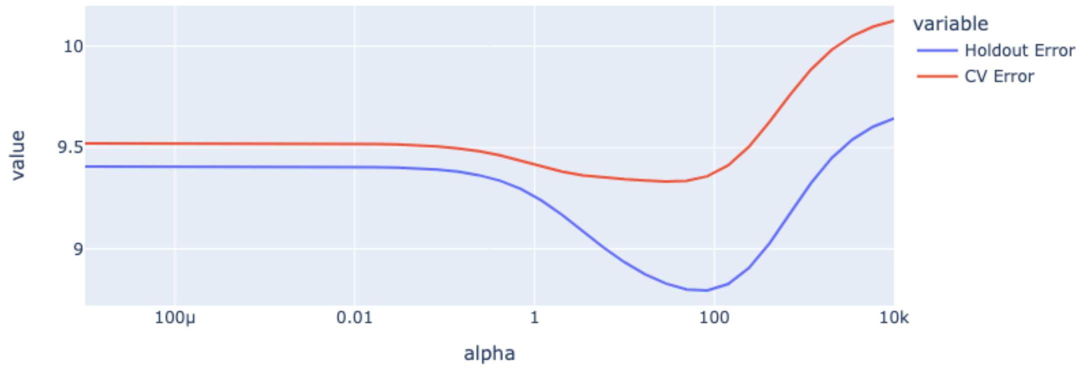
```
        split_Y_train, split_Y_valid = Y_train.iloc[train_idx], Y_train.
↪iloc[valid_idx]

        # Fit the model on the training split
        model.fit(split_X_train,split_Y_train)

        # Compute the RMSE on the validation split
        error = rmse(split_Y_valid, model.predict(split_X_valid))


        validation_errors.append(error)

    return np.mean(validation_errors)
```

[44]:
```
cv_errors = []
range_of_alphas = 10**np.linspace(-5, 4, 40)

for alpha in range_of_alphas:
    model = Ridge(alpha)
    cv_error = compute_CV_error(model, X_train2, Y_train2)
    cv_errors.append(cv_error)

error_vs_alpha["CV Error"] = cv_errors
error_vs_alpha.head()
```

[44]:

	alpha	Training Error	Holdout Error	CV Error
0	0.000010	8.987461	9.406277	9.520060
1	0.000017	8.987461	9.406275	9.520058
2	0.000029	8.987461	9.406273	9.520056
3	0.000049	8.987461	9.406269	9.520052
4	0.000084	8.987461	9.406263	9.520046

[45]:
```
px.line(error_vs_alpha, x = "alpha", y = ["Holdout Error", "CV Error"],␣
↪log_x=True)
```



The cross-validation error corroborates our alpha choice of about 83, corresponding with a choice of about 2 features. The minimum CV error occurs with an alpha value of 83.76.

Now we'll examine the CV error for different candadite models:

[46]:
```
len(candidates_best_mod['Features'])
```

[46]: 8

[47]:
```
def compute_CV_error_thesefeatures(model, features, X_train, Y_train):
    '''
    Split the training data into 4 subsets.
    For each subset,
        fit a model holding out that subset
        compute the MSE on that subset (the validation set)
    You should be fitting 4 models total.
    Return the average MSE of these 4 folds.

    Args:
        model: an sklearn model with fit and predict functions
        X_train (data_frame): Training data
        Y_train (data_frame): Label

    Return:
        the average validation MSE for the 4 splits.
    '''
    kf = KFold(n_splits=4)
    validation_errors = []
    X_train = X_train.loc[:,features]

    for train_idx, valid_idx in kf.split(X_train):
```

```
        # split the data
        split_X_train, split_X_valid = X_train.iloc[train_idx], X_train.
↪iloc[valid_idx]
        split_Y_train, split_Y_valid = Y_train.iloc[train_idx], Y_train.
↪iloc[valid_idx]

        # Fit the model on the training split
        model.fit(split_X_train,split_Y_train)

        # Compute the RMSE on the validation split
        error = rmse(split_Y_valid, model.predict(split_X_valid))


        validation_errors.append(error)

    return np.mean(validation_errors)
```

[48]:
```
cand_features = candidates_best_mod['Features'].reset_index()
```

[49]:
```
cand_features.iloc[1,1]
```

[49]: ['PBLK Grade', 'PFF Grade']

[50]:
```
cv_errors_vs_whichFeatures = pd.DataFrame(columns = ["Features", "CV Error"])
for i in range(0,8):
    cv_error = compute_CV_error_thesefeatures(model = lm.LinearRegression(),␣
↪features = cand_features.iloc[i,1], X_train = X_train1, Y_train=Y_train1)
    cv_errors_vs_whichFeatures.loc[len(cv_errors_vs_whichFeatures)] =␣
↪[cand_features.iloc[i,1], cv_error]
```

[51]:
```
cv_errors_vs_whichFeatures
```

[51]:

	Features	CV Error
0	[PBLK Grade, RBLK Grade]	9.423650
1	[PBLK Grade, PFF Grade]	9.605082
2	[RBLK Grade, PFF Grade]	9.485767
3	[RBLK Grade, EFF]	9.417030
4	[PFF Grade]	9.535513
5	[PFF Grade, EFF]	9.553048
6	[PBLK Grade, RBLK Grade, EFF]	9.509575
7	[PBLK Grade, EFF, PFF Grade]	9.699980

The best CV error was given by the model that used PBLK Grade and RBLK Grade as its features, and the model that used RBLK and PFF Grade as its features (both at ~9.42). This also supports the notion that the alpha of about 83 corresponds to 2 features.

We'll go with RBLK Grade and PBLK Grade as our chosen features, because it corroborates the choice of the holdout error analysis earlier. This is a slight surprise that PBLK Grade is involved in the predictin given that it had the lesser correlation with NFL score (~.2) where RBLK and PFF Grade had about a .4 correlation strength. However, I will trust the results of cross validation error more, considering that each error is the result of an average of several trained models.

[52]:
```
prospects.head()
```

[52]:
```
          Player  PFF Grade  PBLK Grade  RBLK Grade  PEN Rate   EFF Pos  \
0    Ikem Ekwonu      84.30       67.03       90.10      3.30  97.5  OL
1      Evan Neal      80.40       76.23       79.07      2.00  98.6  OL
2  Charles Cross      75.25       77.95       73.75      7.00  97.3  OL
3   Kenyon Green      72.27       56.53       77.20      5.00  97.6  OL
4   Zion Johnson      77.50       74.40       76.70      1.33  98.0  OL

   Drafted  Drafted in Pos College NFL TM  Draft Year  Height  Weight  \
0        6               1    NCST    CAR        2022      76     320
1        7               2     ALA    NYG        2022      79     370
2        9               3  MISSST    SEA        2022      77     305
3       15               4    TA&M    HOU        2022      76     325
4       17               5      BC    LAC        2022      75     316

   NFL_score
0      80.20
1      69.35
2      81.75
3      66.25
4      80.95
```

Adding column to dataframe that contains the final predictions:

[53]:
```
final_predictors = X.loc[:,['PBLK Grade', 'RBLK Grade']]
linear_model.fit(final_predictors, y)
prospects['final_predictions'] = linear_model.predict(final_predictors)
```

1.9 Post-Experimentation EDA

How much more valuable are the earlier draft picks on average? What is the depth of the position in drafts?

[50]:
```
score_by_draftpos = prospects.groupby('Drafted in Pos')[['NFL_score']].agg(np.
 ↪mean)
score_by_draftpos
```

[50]:
```
                NFL_score
Drafted in Pos
1               87.338333
2               86.215000
3               83.283333
4               79.350000
```

```
5                  83.291667
6                  85.008333
7                  84.551667
8                  80.185000
9                  78.280000
10                 78.403333
11                 73.970000
12                 76.540000
13                 78.360000
14                 75.638833
15                 77.443333
16                 81.930000
17                 73.036000
18                 76.021667
19                 66.776667
20                 71.230000
21                 69.481667
22                 76.203333
23                 70.216667
24                 71.886667
25                 73.933333
26                 75.793333
27                 72.481667
28                 78.028333
29                 60.720000
30                 70.962000
```

[53]:
```
score_by_draftpos['NFL_score'][0:9].mean()
```

[53]: 83.05592592592592

[54]:
```
score_by_draftpos['NFL_score'][10:19].mean()
```

[54]: 75.52405555555555

[55]:
```
score_by_draftpos['NFL_score'][20:29].mean()
```

[55]: 72.08277777777779

Evidently, top 10 OL picks have an average NFL score of 83, but then the next 10 have an average of 75, where 20-30 have a 72 avg.

the gap between Top 10 and 10-20 is larger than the gap between 10-20 and 20-30

Which schools produce the most prospects

[78]:
```
freq_college = prospects.groupby('College').size().sort_values(ascending =␣
↪False).to_frame()
```

[79]:
```
freq_college = freq_college.rename({0:'count'},axis=1)
freq_college
```

[79]:
```
         count
College
ALA          6
OKL          6
UCLA         5
WISC         5
FLA          5
…          …
DUKE         1
CTNGA        1
SUTA         1
COLST        1
WY           1

[83 rows x 1 columns]
```

Out of the Schools the produced at least 2 prospects, Which school has given the best players?

[80]:
```
freq_college_g2 = freq_college.query("count > 2").reset_index()
```

[83]:
```
college_by_mean = prospects.groupby('College')[['NFL_score']].mean().
 ↪sort_values(by = 'NFL_score', ascending = False).reset_index()
```

[84]:
```
college_by_mean[college_by_mean['College'].isin(freq_college_g2['College'])]
```

[84]:
```
   College   NFL_score
2     NTRE   90.570000
19   MISSO   83.183333
24   OHIST   82.020000
25    NCST   81.740000
26  MISSST   80.613333
27    WISC   80.542000
29    IOWA   80.157500
31     FLA   79.604000
32     LOU   79.456667
33     MIA   78.870000
38   WMICH   77.710000
40    OREG   76.827667
45     IND   75.870000
47    UCLA   75.610000
48     OKL   75.221667
51     ALA   75.001667
52    OHST   74.900000
```

```
53     TA&M  74.240000
54    WASST  74.023333
58      LSU  73.083333
60      UTA  72.610000
61      USC  72.593333
62      FST  72.396667
63     MISS  72.333333
64      GEO  72.290000
66      TCU  71.115000
68      PIT  69.145000
```

[85]:
```
len(freq_college_g2)
```

[85]:
```
27
```

Which NFL Teams have had the best drafts over the last 7 years?

[86]:
```
NFLTM_by_mean = prospects.groupby('NFL TM')[['NFL_score']].mean().
  ↪sort_values(by = 'NFL_score', ascending = False).reset_index()
```

[87]:
```
NFLTM_by_mean
```

[87]:
```
   NFL TM  NFL_score
0     DET  86.271667
1     IND  85.391429
2     ATL  84.100000
3     BUF  83.537500
4     CAR  80.945000
5      NO  80.630000
6     CLE  80.056667
7     TEN  79.910000
8     BAL  79.491429
9      LV  78.958000
10    CHI  78.872600
11    JAX  78.494000
12    ARI  78.402000
13     NE  77.713333
14    DEN  77.194286
15    PHI  77.152000
16    DAL  76.448000
17     TB  76.066667
18    HOU  75.311667
19    WAS  75.045000
20     KC  74.656667
21    SEA  74.397000
22    NYJ  74.215000
23     SF  74.194000
```

```
24    LAC  74.043333
25    MIN  73.723750
26    MIA  72.780000
27    CIN  72.761250
28     GB  71.636667
29    LAR  71.273333
30    PIT  70.790000
31    NYG  70.348333
32     AZ  60.000000
```

[91]:
```
prospects.loc[:,['Player ', 'NFL_score']].
 ↪sort_values(by='NFL_score',ascending=False)
```

[91]:
```
                  Player   NFL_score
91           Ryan Ramczyk      94.83
149       Chris Lindstrom      93.66
118        Quenton Nelson      92.93
63          Taylor Decker      92.24
55              Shaq Mason      92.18
..                    …          …
115          Kofi Amichia      60.00
29        Marcus McKethan      60.00
88    Christian Westerman      60.00
167             Dru Samia      59.80
146           Jamil Demby      54.40

[177 rows x 2 columns]
```

1.10 Conclusion

After feature engineering and model selection with holdout error, regularization, and cross-validation error, I settled on the model that utilized Pass Blocking and Run Blocking both as predictors.

Evaluation

Ideally, I would've collected data from a couple mopre drafts, like 2020 and 2021, but my time was limited.

Overall, none of the models performed amazingly, but I'm satisfied with the process of figuring out which model most effectively reduced risk in the draft process. Since the final CV error of the model i chose was about 9, this is approximately equal to the standard deviation of NFL score. We can assume that the model will be likely be within 1 SD of the actual NFL score ~68 percent of the time assuming that the distribution of NFL score is approx normal due to Central Limit Theorem.

I would like to include some of my findings in the post-experiment EDA in the model, like maybe adding a 5 percent bump in predicted NFL score to those that are products of the top schools for O-Lineman.

I'm in the process of making an app in which you can see the suggested draft order of 2023 OL prospects based on my model

[ ]:
```

```