Introduction

Group players according to their ability to do damage on contact. Then build individually targeted improvement plans for all of the hitters. Data include pitch level Trackman data from games and swing level Blast Motion data from practice.

Method

All in all, I used cluster analysis on practice data (Blast) and came up with four groups. To make development plans for them, I had to see the characteristics of these three groups and their respective in-game performance. xwOBA for those contact plays is my choice for evaluation.

Exploratory Data Analysis

In [1]:
```
import pandas as pd
df_trackman = pd.read_csv(r'C:\Users\allen\Desktop\Baseball Analytics C
oding Task\SEA\trackman_data.csv')
df_blast = pd.read_csv(r'C:\Users\allen\Desktop\Baseball Analytics Codi
ng Task\SEA\blast_data.csv')
```

I viewed the top 5 rows of the pandas dataframe with the pandas head() method.

In [2]:
```
df_trackman.head()
```

Out[2]:

	Date	Inning	Top	Outs	Balls	Strikes	PitcherId	BatterId	Bats	Throws	...	PlateSide
0	2019-04-30	4	Top	1	0	0	710e55d6	f70b0d82	Right	Right	...	0.085674

	Date	Inning	Top	Outs	Balls	Strikes	PitcherId	BatterId	Bats	Throws	...	PlateSide
1	2019-04-30	4	Top	1	0	1	710e55d6	f70b0d82	Right	Right	...	0.646820
2	2019-04-30	4	Top	1	0	2	710e55d6	f70b0d82	Right	Right	...	0.304825
3	2019-05-06	5	Bottom	0	0	0	bf435272	b4417992	Right	Right	...	1.000660
4	2019-05-06	5	Bottom	0	1	0	bf435272	b4417992	Right	Right	...	0.582892

5 rows × 23 columns

In [3]:
```
df_blast.head()
```

Out[3]:

	BatterId	Date	AttackAngle	BatSpeed	Connection	EarlyConnection	Handedness	PlanarEff
0	2e612ce7	2019-01-02	0.111074	30.490201	1.428424	1.507817	5	0.
1	2e612ce7	2019-01-02	0.222480	29.838648	1.358282	1.442910	5	0.
2	2e612ce7	2019-01-02	0.126757	29.619088	1.339027	1.466272	5	0.
3	2e612ce7	2019-01-02	0.248148	29.013107	1.422598	1.557318	5	0.
4	367fb7f9	2019-01-06	0.149912	31.725814	1.501380	1.344469	5	0.

I viewed the summary of the dataframe with the pandas info() method.

In [4]:
```
df_trackman.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74910 entries, 0 to 74909
```

```
Data columns (total 23 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Date         74910 non-null  object 
 1   Inning       74910 non-null  int64  
 2   Top          74910 non-null  object 
 3   Outs         74910 non-null  int64  
 4   Balls        74910 non-null  int64  
 5   Strikes      74910 non-null  int64  
 6   PitcherId    74910 non-null  object 
 7   BatterId     74910 non-null  object 
 8   Bats         74910 non-null  object 
 9   Throws       74910 non-null  object 
 10  PitchNumber  74910 non-null  int64  
 11  PAofInning   74910 non-null  int64  
 12  PitchofPA    74910 non-null  int64  
 13  PlateSide    74296 non-null  float64
 14  PlateHeight  74296 non-null  float64
 15  ExitSpeed    18227 non-null  float64
 16  VertAngle    18227 non-null  float64
 17  HorzAngle    18227 non-null  float64
 18  HitSpinRate  13723 non-null  float64
 19  PitchType    74910 non-null  object 
 20  PitchCall    74910 non-null  object 
 21  PlayResult   74910 non-null  object 
 22  HitType      74910 non-null  object 
dtypes: float64(6), int64(7), object(10)
memory usage: 13.1+ MB
```

In [5]:
```
df_blast.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109443 entries, 0 to 109442
Data columns (total 9 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   BatterId                 109443 non-null  object 
 1   Date                     109443 non-null  object 
 2   AttackAngle              109443 non-null  float64
```

```
 3   BatSpeed               109443 non-null  float64
 4   Connection             109443 non-null  float64
 5   EarlyConnection        109443 non-null  float64
 6   Handedness             109443 non-null  int64
 7   PlanarEfficiency       109443 non-null  float64
 8   RotationalAcceleration  109443 non-null  float64
dtypes: float64(6), int64(1), object(2)
memory usage: 7.5+ MB
```

I look at the descriptive statistics of the dataframe with the pandas describe() method.

In [6]:
```
df_trackman.describe()
```

Out[6]:

	Inning	Outs	Balls	Strikes	PitchNumber	PAofInning	F
count	74910.000000	74910.000000	74910.000000	74910.000000	74910.000000	74910.000000	7491
mean	4.981298	0.984448	0.888186	0.873048	147.349766	2.888960	
std	2.605596	0.813463	0.971565	0.826324	88.518225	1.643858	
min	1.000000	0.000000	0.000000	0.000000	1.000000	0.000000	
25%	3.000000	0.000000	0.000000	0.000000	72.000000	2.000000	
50%	5.000000	1.000000	1.000000	1.000000	145.000000	3.000000	
75%	7.000000	2.000000	2.000000	2.000000	217.000000	4.000000	
max	15.000000	2.000000	3.000000	2.000000	445.000000	13.000000	2

In [7]:
```
df_blast.describe()
```

Out[7]:

	AttackAngle	BatSpeed	Connection	EarlyConnection	Handedness	PlanarEffici
count	109443.000000	109443.000000	109443.000000	109443.000000	109443.000000	109443.00
mean	0.198839	30.868134	1.420197	1.647542	4.632146	0.68
std	0.132947	2.949614	0.164880	0.262527	0.482223	0.11

	AttackAngle	BatSpeed	Connection	EarlyConnection	Handedness	PlanarEffici
min	-0.966112	13.415024	0.528269	0.543694	4.000000	0.25
25%	0.121744	29.370584	1.308698	1.468902	4.000000	0.60
50%	0.206185	31.254110	1.421837	1.638253	5.000000	0.69
75%	0.284448	32.776712	1.532932	1.821757	5.000000	0.77
max	0.987156	40.152891	2.211674	2.788933	5.000000	0.99

Take a look at the mean value of data from blast with different BatterId

In [8]:
```
df_blast_mean = df_blast.groupby('BatterId').mean()
df_blast_mean = df_blast_mean.drop(['Handedness'], axis=1)
```

In [9]:
```
#df_trackman_bat_mean = df_trackman_bat_direction_adjusted.groupby('Bat
terId').mean()
#df_combine = df_trackman_bat_mean.merge(df_blast_mean, how='inner', on
='BatterId')
#df_combine = df_combine[['AttackAngle', 'BatSpeed', 'Connection', 'Ear
lyConnection', 'PlanarEfficiency', 'RotationalAcceleration']]
```

Clustering the data from blast

Step 1: Reduce Dimensionality

Find the optimal number of components which capture the greatest amount of variance in the data. In my case, as seen in the figure below, that number is three.

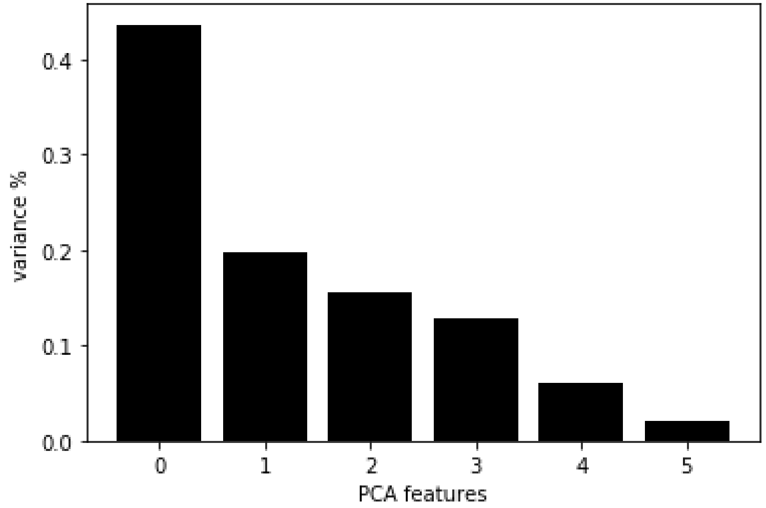
In [10]:
```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

```
X = StandardScaler().fit_transform(df_blast_mean)

# Create a PCA instance: pca
pca = PCA(n_components=6)
principalComponents = pca.fit_transform(X)

# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='black')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)

# Save components to a DataFrame
PCA_components = pd.DataFrame(principalComponents)
```



Step 2: Find the Clusters

In this step, I will use k-means clustering to view the top three PCA components. In order to do this, I will first fit these principal components to the k-means algorithm and determine the best number of clusters. Determining the ideal number of clusters for our k-means model can be done by measuring the sum of the squared distances to the nearest cluster center aka inertia. Much like the scree plot for PCA, the k-means scree plot below indicates the percentage of variance explained, but in slightly different terms, as a function of the number of clusters.

In [11]:
```
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(PCA_components.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

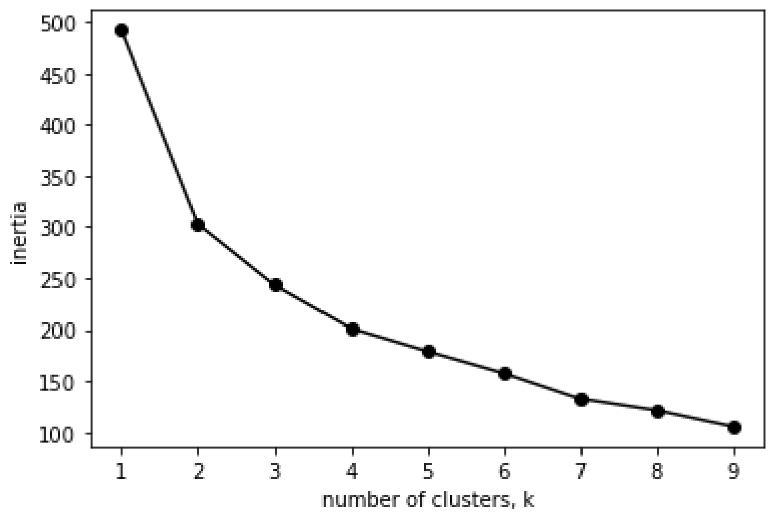


Figure shows that after 4 clusters at (the elbow) the change in the value of inertia is no longer significant and most likely, neither is the variance of the rest of the data after the elbow point. Therefore we can discard everything after k=4 and proceed to the last step in the process.

In [12]:
```
kmeans = KMeans(n_clusters=4).fit(PCA_components.iloc[:,:4])
kmeans.labels_
df_blast_mean['labels'] = kmeans.labels_
```

After having the clustering result, I need to interpret the clusters. The easiest way to describe clusters is by using a set of rules. I could automatically generate the rules by training a decision tree model using original features and clustering result as the label. I wrote a cluster_report function that wraps the decision tree training and rules extraction from the tree.

In [13]:
```
from IPython.display import display, HTML
from sklearn.tree import _tree, DecisionTreeClassifier
import pandas as pd

def pretty_print(df):
    return display( HTML( df.to_html().replace("\\n","<br>") ) )

def get_class_rules(tree: DecisionTreeClassifier, feature_names: list):
  inner_tree: _tree.Tree = tree.tree_
  classes = tree.classes_
  class_rules_dict = dict()

  def tree_dfs(node_id=0, current_rule=[]):
    # feature[i] holds the feature to split on, for the internal node
 i.
    split_feature = inner_tree.feature[node_id]
    if split_feature != _tree.TREE_UNDEFINED: # internal node
      name = feature_names[split_feature]
      threshold = inner_tree.threshold[node_id]
      # left child
      left_rule = current_rule + ["({} <= {})".format(name, threshold)]
      tree_dfs(inner_tree.children_left[node_id], left_rule)
      # right child
      right_rule = current_rule + ["({} > {})".format(name, threshold)]
```

```
      tree_dfs(inner_tree.children_right[node_id], right_rule)
    else: # leaf
      dist = inner_tree.value[node_id][0]
      dist = dist/dist.sum()
      max_idx = dist.argmax()
      if len(current_rule) == 0:
        rule_string = "ALL"
      else:
        rule_string = " and ".join(current_rule)
      # register new rule to dictionary
      selected_class = classes[max_idx]
      class_probability = dist[max_idx]
      class_rules = class_rules_dict.get(selected_class, [])
      class_rules.append((rule_string, class_probability))
      class_rules_dict[selected_class] = class_rules

  tree_dfs() # start from root, node_id = 0
  return class_rules_dict

def cluster_report(data: pd.DataFrame, clusters, min_samples_leaf=50, p
runing_level=0.01):
    # Create Model
    tree = DecisionTreeClassifier(min_samples_leaf=min_samples_leaf, cc
p_alpha=pruning_level)
    tree.fit(data, clusters)

    # Generate Report
    feature_names = data.columns
    class_rule_dict = get_class_rules(tree, feature_names)

    report_class_list = []
    for class_name in class_rule_dict.keys():
        rule_list = class_rule_dict[class_name]
        combined_string = ""
        for rule in rule_list:
            combined_string += "[{}] {}\n\n".format(rule[1], rule[0])
        report_class_list.append((class_name, combined_string))

    cluster_instance_df = pd.Series(clusters).value_counts().reset_inde
x()
```

```
    cluster_instance_df.columns = ['class_name', 'instance_count']
    report_df = pd.DataFrame(report_class_list, columns=['class_name',
'rule_list'])
    report_df = pd.merge(cluster_instance_df, report_df, on='class_nam
e', how='left')
    pretty_print(report_df.sort_values(by='class_name')[['class_name',
'instance_count', 'rule_list']])
```

Take a look at the report generated.

In [14]:
```
cluster_report(df_blast_mean.drop(['labels'], axis=1), df_blast_mean['l
abels'], min_samples_leaf=15, pruning_level=0.01)
```

	class_name	instance_count	rule_list
3	0	18	[0.8666666666666667] (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (RotationalAcceleration <= 120.16216659545898)
1	1	27	[0.95] (EarlyConnection <= 1.7521717548370361) and (AttackAngle > 0.1938438042998314) and (EarlyConnection <= 1.6067730784416199)
0	2	36	[0.47058823529411764] (EarlyConnection <= 1.7521717548370361) and (AttackAngle > 0.1938438042998314) and (EarlyConnection > 1.6067730784416199) [0.9629629629629629] (EarlyConnection > 1.7521717548370361)
2	3	23	[0.84] (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (RotationalAcceleration > 120.16216659545898)

Build my own classification model for xwOBA

To properly examine the game performance for these three groups of batters, I decide to come up with xwOBA value using the data I have from trackman

Take a look at what I have

In [15]:
```
df_trackman.PitchCall.unique()
```

Out[15]:
```
array(['StrikeCalled', 'FoulBall', 'BallCalled', 'StrikeSwinging',
       'InPlay', 'HitByPitch', 'BallIntentional', 'Undefined',
       'CatchersInterference'], dtype=object)
```

In [16]:
```
df_trackman.PlayResult.unique()
```

Out[16]:
```
array(['Undefined', 'Out', 'Single', 'Sacrifice', 'Double', 'Triple',
       'HomeRun', 'Error', 'FieldersChoice'], dtype=object)
```

wOBA only considers balls in play, walk and hitbypitch; similarly, xwOBA considers the probability of each event using exit velocity and launch angle.

In [17]:
```
#filter to wanted columns
event_include = ['BallCalled', 'InPlay', 'HitByPitch']
df_trackman_xwOBA = df_trackman[df_trackman['PitchCall'].isin(event_inc
lude)]

#assign hitbypitch, walk to column 'PlayResult'
df_trackman_xwOBA.loc[df_trackman_xwOBA['PitchCall']=='HitByPitch', 'Pl
ayResult'] = 'HitByPitch'
df_trackman_xwOBA.loc[(df_trackman_xwOBA['PitchCall']=='BallCalled') &
(df_trackman_xwOBA['Balls']==3), 'PlayResult'] = 'Walk'
df_trackman_xwOBA = df_trackman_xwOBA.drop(df_trackman_xwOBA[df_trackma
n_xwOBA.PlayResult=='Undefined'].index)

#any long-version out = out
outs = ['Out', 'Sacrifice', 'Error', 'FieldersChoice']
df_trackman_xwOBA.loc[df_trackman_xwOBA['PlayResult'].isin(outs), 'Play
Result'] = 'Out'

# verify remaining outcomes
df_trackman_xwOBA['PlayResult'].unique()
```

```
C:\Users\allen\anaconda3\lib\site-packages\pandas\core\indexing.py:965:

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
  self.obj[item] = s
```

Out[17]:
```
array(['Out', 'Walk', 'Single', 'Double', 'Triple', 'HomeRun',
       'HitByPitch'], dtype=object)
```

Now that I have simplified plate-appearance outcomes, I'll join in Fangraphs' wOBA values for the given season(2019).

In [18]:
```
woba_weights = pd.read_csv(r'C:\Users\allen\Desktop\Baseball Analytics
 Coding Task\SEA\woba_weights.csv')
woba_weights = woba_weights.loc[woba_weights['Season']==2019, ['wBB',
'wHBP', 'w1B', 'w2B', 'w3B', 'wHR']]
woba_weights
```

Out[18]:

	wBB	wHBP	w1B	w2B	w3B	wHR
2	0.69	0.719	0.87	1.217	1.529	1.94

Assign the values to my dataframe 'df_trackman_xwOBA'

In [19]:
```
df_trackman_xwOBA['wBB']=0.69
df_trackman_xwOBA['wHBP']=0.719
df_trackman_xwOBA['w1B']=0.87
df_trackman_xwOBA['w2B']=1.217
df_trackman_xwOBA['w3B']=1.529
df_trackman_xwOBA['wHR']=1.94
```

Build the models from rows with actual exit velocity, launch angle values

In [20]:
```
df_trackman_xwOBA_contact_known = df_trackman_xwOBA[df_trackman_xwOBA[
'ExitSpeed'].notnull()]
event_include = ['Single', 'Out', 'Double', 'Triple', 'HomeRun']
df_trackman_xwOBA_contact_known = df_trackman_xwOBA_contact_known[df_tr
ackman_xwOBA_contact_known['PlayResult'].isin(event_include)]
```

My goal here isn't necessarily to predict the outcome of a hit as accurately as possible.

If I'm trying to uncover a hitter's true talent, I'll build models using only the things the hitter is responsible for:

batted ball speed

batted ball vertical angle (launch angle)

batted ball horizontal angle (spray angle)

handedness (to standardize spray angle)

As far as the models themselves go, I mostly care about the probabilistic predictions from each model. I can get the outcome classification from that data, but more importantly, those probabilities are useful. If we assign a value to the results of a batted ball, we can calculate the expected value of the batted ball and use that to value a hitter.

I've settled on 6 popular classifiers to compare. I'll use:

logistic regression k-nearest neighbors support vector machine decision tree random forest gradient boosting

And I'm going to use 4 metrics to evaluate the models, which together should give a good picture of the best overall model:

F1 score (weighted by instances of each label) ROC AUC (computed by label and weighted by frequency) balanced accuracy (for imbalanced datasets) log loss I'll run with largely default settings for each of the models to keep a relatively level playing field.

In [21]:
```
df_trackman_xwOBA_contact_known

# one-hot encode handedness
df_trackman_xwOBA_contact_known = pd.concat([df_trackman_xwOBA_contact_
known, pd.get_dummies(df_trackman_xwOBA_contact_known.Bats)], axis=1)

# drop unnecessary columns & rename to be a little clearer
df_trackman_xwOBA_contact_known = df_trackman_xwOBA_contact_known.drop(
columns=['Left', 'S'])
df_trackman_xwOBA_contact_known = df_trackman_xwOBA_contact_known.renam
e(columns={'Right': 'is_Right'})
```

In [22]:
```
#select the variables I want to include
df_trackman_xwOBA_contact_known_model = df_trackman_xwOBA_contact_known
[['ExitSpeed', 'VertAngle', 'HorzAngle', 'is_Right', 'PlayResult']]

#scale the numeric data
to_scale = ['ExitSpeed', 'VertAngle', 'HorzAngle']
df_trackman_xwOBA_contact_known_model[to_scale] = StandardScaler().fit_
transform(df_trackman_xwOBA_contact_known_model[to_scale])

#assign x and y for my model
X = df_trackman_xwOBA_contact_known_model[['ExitSpeed', 'VertAngle', 'H
orzAngle', 'is_Right']]
y = df_trackman_xwOBA_contact_known_model['PlayResult']
```

```
C:\Users\allen\anaconda3\lib\site-packages\ipykernel_launcher.py:6: Set
tingWithCopyWarning:
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

C:\Users\allen\anaconda3\lib\site-packages\pandas\core\indexing.py:965:
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
  self.obj[item] = s
```

In [23]:

```python
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from sklearn.metrics import f1_score, accuracy_score, log_loss, roc_auc_score, make_scorer
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

# scoring metrics
scoring = {
    'f1_weighted': 'f1_weighted',
    'accuracy': 'balanced_accuracy',
    'roc_auc': 'roc_auc_ovr_weighted',
    'neg_log_loss': 'neg_log_loss'
    }

# for results df
eval_cols = [
    'models',
    'F1 Score',
    'Balanced Accuracy',
    'ROC AUC',
    'Neg Log Loss'
    ]

# define classifier models
classifiers = [
    LogisticRegression(multi_class='multinomial'),
    KNeighborsClassifier(),
```

```
    SVC(probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    GradientBoostingClassifier(),
    XGBClassifier()
    ]

# classifier names
clf_names = [
    'Logistic Regression',
    'KNN',
    'SVM',
    'Decision Tree',
    'Random Forest',
    'Gradient Boosting',
    'XGBClassifier'
    ]
```

```
C:\Users\allen\anaconda3\lib\site-packages\sklearn\externals\six.py:31:
FutureWarning: The module is deprecated in version 0.21 and will be rem
oved in version 0.23 since we've dropped support for Python 2.7. Please
rely on the official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
C:\Users\allen\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:144: FutureWarning: The sklearn.neighbors.base module is  deprecated
in version 0.22 and will be removed in version 0.24. The corresponding
classes / functions should instead be imported from sklearn.neighbors.
Anything that cannot be imported from sklearn.neighbors is now part of
the private API.
  warnings.warn(message, FutureWarning)
```

In [24]:

```
import time as time
import numpy as np
f1, acc, roc_auc, log_loss = [], [], [], []
for clf, clf_nm in zip(classifiers, clf_names):

    start = time.time()

    # cross-validate 5 times
```

```
    res = cross_validate(clf, X, y, cv=5, scoring=scoring)
    results = pd.DataFrame(res)

    stop = time.time()

    print('Time to cross-validate %s = %0.3f min.' % (clf_nm, (stop - s
tart) / 60))

    # save average scores
    f1.append(np.mean(results.test_f1_weighted))
    acc.append(np.mean(results.test_accuracy))
    roc_auc.append(np.mean(results.test_roc_auc))
    log_loss.append(np.mean(results.test_neg_log_loss))

# save results to df
model_eval = pd.DataFrame(data=zip(clf_names, f1, acc, roc_auc, log_los
s),
                          columns=eval_cols)

display(model_eval)
```

```
Time to cross-validate Logistic Regression = 0.071 min.
Time to cross-validate KNN = 0.042 min.
Time to cross-validate SVM = 2.581 min.
Time to cross-validate Decision Tree = 0.012 min.
Time to cross-validate Random Forest = 0.284 min.
Time to cross-validate Gradient Boosting = 1.080 min.
Time to cross-validate XGBClassifier = 0.336 min.
```

	models	F1 Score	Balanced Accuracy	ROC AUC	Neg Log Loss
0	Logistic Regression	0.510787	0.264336	0.668918	-0.877740
1	KNN	0.751605	0.508681	0.853699	-2.626078
2	SVM	0.731226	0.479989	0.852804	-0.631829
3	Decision Tree	0.694244	0.474296	0.716908	-10.597652
4	Random Forest	0.754276	0.511131	0.875267	-0.843073
5	Gradient Boosting	0.751723	0.497045	0.876578	-0.610547

	models	F1 Score	Balanced Accuracy	ROC AUC	Neg Log Loss
6	XGBClassifier	0.743082	0.481209	0.873540	-0.600397

Overall, the XGBoost model was the best. Thus I make the prediction using it.

In [25]:

```
model=XGBClassifier()
model.fit(X, y)
hit_probs = pd.DataFrame(model.predict_proba(X), columns=model.classes_
)
hit_probs
```

Out[25]:

	Double	HomeRun	Out	Single	Triple
0	0.013125	0.002463	0.405336	0.576670	0.002406
1	0.006197	0.001341	0.679590	0.311756	0.001116
2	0.278198	0.001274	0.556532	0.161490	0.002505
3	0.022775	0.001588	0.413983	0.559204	0.002450
4	0.008117	0.001791	0.854845	0.133951	0.001296
...	...	...	...	...	...
10838	0.115632	0.309597	0.542037	0.018721	0.014012
10839	0.091019	0.014651	0.712510	0.164413	0.017406
10840	0.034326	0.003106	0.411727	0.547865	0.002976
10841	0.068506	0.002443	0.093785	0.799056	0.036209
10842	0.039520	0.001727	0.454324	0.501577	0.002851

10843 rows × 5 columns

Separate the df_trackman_xwOBA into three:

1. df_trackman_xwOBA_contact_known: the one that I built earlier for those contact plays with exit velo, launch angle data

2. df_trackman_xwOBA_contact_unknown: those contct plays without exit velo, launch angle data
3. df_trackman_xwOBA_noncontact: non-contact plays

In [26]:
```
df_trackman_xwOBA_contact = df_trackman_xwOBA[df_trackman_xwOBA['PlayResult'].isin(['Out', 'Single', 'Double', 'Triple', 'HomeRun'])]
df_trackman_xwOBA_contact_unknown = df_trackman_xwOBA_contact[df_trackman_xwOBA_contact.ExitSpeed.isnull()]
df_trackman_xwOBA_noncontact = df_trackman_xwOBA[df_trackman_xwOBA['PlayResult'].isin(['Walk', 'HitByPitch'])]
```

Minor adjustments to join the tables together

In [27]:
```
df_trackman_xwOBA_contact_known = df_trackman_xwOBA_contact_known.reset_index()
df_trackman_xwOBA_contact_known[['Double', 'HomeRun', 'Out', 'Single', 'Triple']] = hit_probs
df_trackman_xwOBA_contact_known = df_trackman_xwOBA_contact_known.drop(columns=['index'])
df_trackman_xwOBA_noncontact['Double']=np.zeros(len(df_trackman_xwOBA_noncontact))
df_trackman_xwOBA_noncontact['HomeRun']=np.zeros(len(df_trackman_xwOBA_noncontact))
df_trackman_xwOBA_noncontact['Out']=np.zeros(len(df_trackman_xwOBA_noncontact))
df_trackman_xwOBA_noncontact['Single']=np.zeros(len(df_trackman_xwOBA_noncontact))
df_trackman_xwOBA_noncontact['Triple']=np.zeros(len(df_trackman_xwOBA_noncontact))
df_trackman_xwOBA_combine = pd.concat([df_trackman_xwOBA_contact_known, df_trackman_xwOBA_noncontact])

# add marker for ball in play
df_trackman_xwOBA_combine['contact'] = np.zeros(len(df_trackman_xwOBA_combine))
df_trackman_xwOBA_combine.loc[df_trackman_xwOBA_combine['Double']!=0, 'contact'] = 1
```

```
C:\Users\allen\anaconda3\lib\site-packages\ipykernel_launcher.py:4: Set
tingWithCopyWarning:
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
  after removing the cwd from sys.path.
C:\Users\allen\anaconda3\lib\site-packages\ipykernel_launcher.py:5: Set
tingWithCopyWarning:
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
  """
C:\Users\allen\anaconda3\lib\site-packages\ipykernel_launcher.py:6: Set
tingWithCopyWarning:
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

C:\Users\allen\anaconda3\lib\site-packages\ipykernel_launcher.py:7: Set
tingWithCopyWarning:
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
  import sys
C:\Users\allen\anaconda3\lib\site-packages\ipykernel_launcher.py:8: Set
tingWithCopyWarning:
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
```

Now I'll write two functions: one for determing the xwOBA value of a PA, and one for determing the wOBA value of a PA.

In [28]:
```
def calc_xwoba(data):
    '''
    Calculate the xwOBA value for a plate appearance. If PA ends on a b
all put in play,
    use hit probabilities to calculate expected wOBA. Else, use known w
OBA value.

    '''

    if data['contact'] == 1:
        xwoba = (data['Single'] * data['w1B'] + data['Double'] * data[
'w2B'] +
                 data['Triple'] * data['w3B'] + data['HomeRun'] * data[
'wHR'])

    elif data['PlayResult'] == 'Walk':
        xwoba = data['wBB']

    elif data['PlayResult'] == 'HitByPitch':
        xwoba = data['wHBP']

    return round(xwoba, 3)


def calc_woba(data):
    '''
    Calculate the wOBA value for a plate appearance. Use the known wOBA
value for each outcome.

    '''

    if data['PlayResult'] == 'Single':
```

```
        woba = data['w1B']

    elif data['PlayResult'] == 'Double':
        woba = data['w2B']

    elif data['PlayResult'] == 'Triple':
        woba = data['w3B']

    elif data['PlayResult'] == 'HomeRun':
        woba = data['wHR']

    elif data['PlayResult'] == 'Walk':
        woba = data['wBB']

    elif data['PlayResult'] == 'HitByPitch':
        woba = data['wHBP']

    else:
        woba = 0

    return round(woba, 3)
```

In [29]:
```
# calculate xwOBA and wOBA for each PA
df_trackman_xwOBA_combine['xwoba'] = df_trackman_xwOBA_combine.apply(calc_xwoba, axis=1)
df_trackman_xwOBA_combine['woba'] = df_trackman_xwOBA_combine.apply(calc_woba, axis=1)
```

Take a look at my df_blast_mean dataframe again

In [30]:
```
df_blast_mean
```

Out[30]:

	AttackAngle	BatSpeed	Connection	EarlyConnection	PlanarEfficiency	RotationalAccel
BatterId						
002a3a2c	0.143681	31.354873	1.252272	1.285903	0.802534	192.

	AttackAngle	BatSpeed	Connection	EarlyConnection	PlanarEfficiency	RotationalAccel
BatterId						
02923b59	0.101737	30.133884	1.375228	1.617774	0.591748	58.!
0325748c	0.146599	31.630133	1.308664	1.363582	0.706607	138.!
0fa51742	0.167412	31.143447	1.546772	1.840998	0.656088	148.:
121483c1	0.069697	30.940481	1.410839	1.589855	0.792978	151.!
...	...	...	...	...	...	
f70b0d82	0.197305	32.519340	1.315247	1.603070	0.704143	115.(
f7985ef1	0.193293	28.549342	1.246320	1.318045	0.802601	97.·
f8c3e062	0.204387	30.358646	1.255127	1.498686	0.714045	102.:
f98aa01e	0.188774	29.696390	1.338029	1.492144	0.694040	86.!
fb7e9a26	0.133913	28.648303	1.266849	1.392805	0.780506	98.:

104 rows × 7 columns

Combine the practice data (df_blast_mean) and in-game data (df_trackman_xwOBA_combine) together

In [31]:

```
df_trackman_xwOBA_combine = df_trackman_xwOBA_combine.merge(df_blast_me
an, how='left', on='BatterId')
df_trackman_xwOBA_combine
```

Out[31]:

	Date	Inning	Top	Outs	Balls	Strikes	PitcherId	BatterId	Bats	Throws	...	conta
0	2019-05-03	8	Bottom	1	2	2	67392fed	b4417992	Right	Right	...	1
1	2019-04-13	8	Bottom	1	0	0	be3a7aca	367fb7f9	Right	Left	...	1
2	2019-05-07	2	Bottom	0	1	0	b1b82ec8	b4417992	Right	Left	...	1

	Date	Inning	Top	Outs	Balls	Strikes	PitcherId	BatterId	Bats	Throws	...	conta
3	2019-04-10	2	Bottom	0	0	2	437d8c83	741921ec	Right	Left	...	1
4	2019-05-17	3	Bottom	0	2	0	245b80b8	b4417992	Right	Right	...	1
...	...	...	...	...	...	...	...	...	...	...	...	
12792	2019-08-15	1	Bottom	1	3	0	ef1db951	5070f997	Left	Right	...	(
12793	2019-08-15	2	Bottom	2	3	2	ef1db951	38598587	Left	Right	...	(
12794	2019-08-15	5	Bottom	0	3	1	ef1db951	38598587	Left	Right	...	(
12795	2019-08-15	2	Bottom	0	3	0	ef1db951	e28cf85c	Left	Right	...	(
12796	2019-08-15	9	Bottom	2	3	0	5fc43dc2	38598587	Left	Right	...	(

12797 rows × 45 columns

In [32]:
```
df_trackman_xwOBA_combine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12797 entries, 0 to 12796
Data columns (total 45 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date                      12797 non-null  object
 1   Inning                    12797 non-null  int64
 2   Top                       12797 non-null  object
 3   Outs                      12797 non-null  int64
 4   Balls                     12797 non-null  int64
 5   Strikes                   12797 non-null  int64
 6   PitcherId                 12797 non-null  object
 7   BatterId                  12797 non-null  object
 8   Bats                      12797 non-null  object
```

```
 9   Throws                      12797 non-null  object 
 10  PitchNumber                 12797 non-null  int64  
 11  PAofInning                  12797 non-null  int64  
 12  PitchofPA                   12797 non-null  int64  
 13  PlateSide                   12783 non-null  float64
 14  PlateHeight                 12783 non-null  float64
 15  ExitSpeed                   10883 non-null  float64
 16  VertAngle                   10883 non-null  float64
 17  HorzAngle                   10883 non-null  float64
 18  HitSpinRate                 7878 non-null   float64
 19  PitchType                   12797 non-null  object 
 20  PitchCall                   12797 non-null  object 
 21  PlayResult                  12797 non-null  object 
 22  HitType                     12797 non-null  object 
 23  wBB                         12797 non-null  float64
 24  wHBP                        12797 non-null  float64
 25  w1B                         12797 non-null  float64
 26  w2B                         12797 non-null  float64
 27  w3B                         12797 non-null  float64
 28  wHR                         12797 non-null  float64
 29  is_Right                    10843 non-null  float64
 30  Double                      12797 non-null  float64
 31  HomeRun                     12797 non-null  float64
 32  Out                         12797 non-null  float64
 33  Single                      12797 non-null  float64
 34  Triple                      12797 non-null  float64
 35  contact                     12797 non-null  float64
 36  xwoba                       12797 non-null  float64
 37  woba                        12797 non-null  float64
 38  AttackAngle                 12797 non-null  float64
 39  BatSpeed                    12797 non-null  float64
 40  Connection                  12797 non-null  float64
 41  EarlyConnection             12797 non-null  float64
 42  PlanarEfficiency            12797 non-null  float64
 43  RotationalAcceleration      12797 non-null  float64
 44  labels                      12797 non-null  int32  
dtypes: float64(27), int32(1), int64(7), object(10)
memory usage: 4.4+ MB
```

Analysis

All in all, I used cluster analysis on practice data (blast) and come up with three groups. To make development plans for them, I have to see the characteristics of these three groups and their respective in-game performance. xwOBA for those contact plays is my choice to evaluate their performance.

In [33]:
```
df_trackman_xwOBA_combine_contact = df_trackman_xwOBA_combine[df_trackm
an_xwOBA_combine.ExitSpeed.notnull()]
df_trackman_xwOBA_combine_contact.groupby('labels').mean()['xwoba']
```

Out[33]:
```
labels
0    0.309546
1    0.398724
2    0.378648
3    0.423701
Name: xwoba, dtype: float64
```

I take a look at wOBA as well to not only evaluate their performance, but also check if my classification models is effective. Judging from similar numbers for xwOBA and wOBA, it seems alright.

In [34]:
```
df_trackman_xwOBA_combine_contact.groupby('labels').mean()['woba']
```

Out[34]:
```
labels
0    0.301480
1    0.405644
2    0.369324
3    0.421520
Name: woba, dtype: float64
```

There are two parameters that we can adjust: min_samples_leaf and pruning_level. Those parameters are controlling the decision tree complexity. To get a more general rule, we could increase the value of min_samples_leaf or pruning_level. Otherwise, if we want to get a more detail rule, we could decrease the value of min_samples_leaf or pruning_level.

```
In [35]: cluster_report(df_blast_mean.drop(['labels'], axis=1), df_blast_mean['labels'], min_samples_leaf=15, pruning_level=0.01)
```

	class_name	instance_count	rule_list
3	0	18	[0.8666666666666667] (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (BatSpeed <= 31.03825283050537)
1	1	27	[0.95] (EarlyConnection <= 1.7521717548370361) and (AttackAngle > 0.1938438042998314) and (EarlyConnection <= 1.6067730784416199)
0	2	36	[0.47058823529411764] (EarlyConnection <= 1.7521717548370361) and (AttackAngle > 0.1938438042998314) and (EarlyConnection > 1.6067730784416199) [0.9629629629629629] (EarlyConnection > 1.7521717548370361)
2	3	23	[0.84] (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (BatSpeed > 31.03825283050537)

```
In [36]: cluster_report(df_blast_mean.drop(['labels'], axis=1), df_blast_mean['labels'], min_samples_leaf=5, pruning_level=0.01)
```

	class_name	instance_count	rule_list
3	0	18	[1.0] (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (BatSpeed <= 30.30543613433838) [0.6] (EarlyConnection <= 1.7521717548370361) and (AttackAngle > 0.1938438042998314) and (PlanarEfficiency > 0.6671400368213654) and (BatSpeed <= 30.50994300842285)
1	1	27	[1.0] (EarlyConnection <= 1.7521717548370361) and (AttackAngle > 0.1938438042998314) and (PlanarEfficiency > 0.6671400368213654) and (BatSpeed > 30.50994300842285)

	class_name	instance_count	rule_list
0	2	36	[0.4] (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (BatSpeed > 30.30543613433838) and (Connection > 1.4180501103401184) [0.8] (EarlyConnection <= 1.7521717548370361) and (AttackAngle > 0.1938438042998314) and (PlanarEfficiency <= 0.6671400368213654) [0.9629629629629629] (EarlyConnection > 1.7521717548370361)
2	3	23	[0.9545454545454546] (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (BatSpeed > 30.30543613433838) and (Connection <= 1.4180501103401184)

Conclusion

Batters in cluster 3 seem to have the best performance. When we look at their clustering feature, (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314) and (BatSpeed > 31.03825283050537) are the general rules. Maintaining those would be my first suggestion. At the same time, they could try on different hitting strategies to see if the variation in other Blast Motion data can lead to better performance. But I think it is more case by case and individuals should focus on maintaining the three features when trying to make minor adjustments.

For a better performance, I would suggest batters in cluster 0, who already possess the features (EarlyConnection <= 1.7521717548370361) and (AttackAngle <= 0.1938438042998314), to work on BatSpeed. I would be really curious about how they perform in games if they push their BatSpeed over the threshold 31. Although having the worst xwOBA value in games, they are actually not that far away from cluster 3 Blast Motion-wise.

As for batters in cluster 1, they already possess similar EarlyConnection and BatSpeed as those in cluster 4. Decreasing their AttackAngle could be beneficial for them to catch players in cluster 3.

As for batters in cluster 2, they could start by decreasing PlanarEfficiency to catch players in cluster 1, since they already possess similar EarlyConnection and AttackAngle. Decreasing their AttackAngle could be their next step.

All in all, this project provides general rules for players to make adjustments with their Blast Motion data in order for a better performance on the field.