In [1]:
```
import pandas as pd
pd.options.display.float_format = '{:.3f}'.format
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import itertools

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import confusion_matrix, classification_report, plot_confusi
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_sco
from sklearn.metrics import roc_curve, auc
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import GridSearchCV
%matplotlib inline
```

In [2]:
```
mlb_df = pd.read_pickle('final_df.pkl')
mlb_df.drop('level_0', axis=1, inplace=True)
mlb_df.set_index('playerID', inplace=True)
mlb_df.head(25)
```

Out[2]:

playerID	g	ab	r	h	2b	3b	hr	rbi	sb	bb	...	k_percentage
aaronha01	3298	12364	2174	3771	624	98	755	2297.000	240.000	1402	...	9.860
abbated01	827	2942	346	748	95	43	11	310.000	138.000	281	...	nan
abbotku01	702	2044	273	523	109	23	62	242.000	22.000	133	...	24.950
abreubo01	2425	8480	1453	2470	574	59	288	1363.000	400.000	1476	...	18.207
abreujo02	901	3547	483	1038	218	14	179	611.000	10.000	245	...	20.257
ackledu01	635	2125	261	512	94	18	46	216.000	31.000	194	...	17.600
adairje01	1165	4019	378	1022	163	19	57	366.000	29.000	208	...	11.900
adamsbo03	1281	4019	591	1082	188	49	37	303.000	67.000	414	...	9.800
adamsbu01	576	2003	282	532	96	12	50	249.000	12.000	234	...	nan
adamsgl01	661	1617	152	452	79	5	34	225.000	6.000	111	...	nan

In [3]:
```
mlb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2316 entries, aaronha01 to zuninmi01
Data columns (total 37 columns):
g                            2316 non-null int64
ab                           2316 non-null int64
r                            2316 non-null int64
h                            2316 non-null int64
2b                           2316 non-null int64
3b                           2316 non-null int64
hr                           2316 non-null int64
rbi                          2316 non-null float64
sb                           2316 non-null float64
bb                           2316 non-null int64
so                           2316 non-null float64
asg_mvp                      2316 non-null float64
baberuth_award               2316 non-null float64
baseball_magazine_allstar    2316 non-null float64
comeback_poy                 2316 non-null float64
gold_glove_award             2316 non-null float64
hankaaron_award              2316 non-null float64
hutch_award                  2316 non-null float64
lougehrig_award              2316 non-null float64
mvp                          2316 non-null float64
nlcs_mvp                     2316 non-null float64
robertoclemente_award        2316 non-null float64
roy                          2316 non-null float64
silver_slugger               2316 non-null float64
tsn_allstar                  2316 non-null float64
triple_crown                 2316 non-null float64
ws_mvp                       2316 non-null float64
k_percentage                 1462 non-null float64
bb_percentage                1462 non-null float64
ba                           1462 non-null float64
slg_percent                  1462 non-null float64
obp                          1462 non-null float64
ops                          1462 non-null float64
iso                          1462 non-null float64
tb                           1462 non-null float64
gidp                         1462 non-null float64
inducted_y                   145 non-null float64
dtypes: float64(29), int64(8)
memory usage: 687.6+ KB
```

In [4]:
```
mlb_df.describe()
```

Out[4]:

	g	ab	r	h	2b	3b	hr	rbi	sb
count	2316.000	2316.000	2316.000	2316.000	2316.000	2316.000	2316.000	2316.000	2316.000
mean	1256.773	4262.729	587.518	1164.007	205.222	37.378	105.435	545.498	88.886
std	534.119	2072.247	347.234	623.185	118.989	31.603	108.079	352.668	106.721
min	420.000	789.000	95.000	182.000	26.000	0.000	0.000	56.000	0.000
25%	847.750	2664.750	326.000	688.000	114.000	16.000	29.000	285.000	22.000
50%	1177.000	3908.000	506.500	1046.000	180.000	28.000	72.000	454.500	51.000
75%	1568.000	5433.000	755.250	1494.000	266.000	49.000	140.000	706.000	118.000
max	3562.000	14053.000	2295.000	4256.000	792.000	302.000	762.000	2297.000	1406.000

8 rows × 37 columns

Fill nulls w/ mean vs. % of each column (Live code switch)

option 1

In [5]:
```
# for column in mlb_df[['k_percentage','bb_percentage','ba','slg_percent','obp','
#    columnSeriesObj = mlb_df[column]
#    print(column)
```

In [6]:
```
mlb_df.bb_percentage.value_counts(normalize=True)
```

Out[6]:
```
6.300    0.008
10.400   0.005
9.600    0.005
8.100    0.005
7.500    0.005
          ...
12.033   0.001
3.200    0.001
7.929    0.001
12.144   0.001
11.850   0.001
Name: bb_percentage, Length: 1033, dtype: float64
```

In [7]:
```
# getting unique values and associated probabilites of each value.
options  = mlb_df.k_percentage.value_counts().index.to_list()
percents = mlb_df.k_percentage.value_counts(normalize=True).to_list()

options1  = mlb_df.bb_percentage.value_counts().index.to_list()
percents1 = mlb_df.bb_percentage.value_counts(normalize=True).to_list()

options2  = mlb_df.ba.value_counts().index.to_list()
percents2 = mlb_df.ba.value_counts(normalize=True).to_list()

options3  = mlb_df.slg_percent.value_counts().index.to_list()
percents3 = mlb_df.slg_percent.value_counts(normalize=True).to_list()

options4  = mlb_df.obp.value_counts().index.to_list()
percents4 = mlb_df.obp.value_counts(normalize=True).to_list()

options5  = mlb_df.ops.value_counts().index.to_list()
percents5 = mlb_df.ops.value_counts(normalize=True).to_list()

options6  = mlb_df.iso.value_counts().index.to_list()
percents6 = mlb_df.iso.value_counts(normalize=True).to_list()

options7  = mlb_df.tb.value_counts().index.to_list()
percents7 = mlb_df.tb.value_counts(normalize=True).to_list()

options8  = mlb_df.gidp.value_counts().index.to_list()
percents8 = mlb_df.gidp.value_counts(normalize=True).to_list()
```

In [8]:
```
#using np.random.choice to select
mlb_df['k_percentage'] = mlb_df['k_percentage'].apply(lambda x: np.random.choice(
mlb_df['bb_percentage'] = mlb_df['bb_percentage'].apply(lambda x: np.random.choic
mlb_df['ba'] = mlb_df['ba'].apply(lambda x: np.random.choice(options2,1, True,per
mlb_df['slg_percent'] = mlb_df['slg_percent'].apply(lambda x: np.random.choice(op
mlb_df['obp'] = mlb_df['obp'].apply(lambda x: np.random.choice(options4,1, True,p
mlb_df['ops'] = mlb_df['ops'].apply(lambda x: np.random.choice(options5,1, True,p
mlb_df['iso'] = mlb_df['iso'].apply(lambda x: np.random.choice(options6,1, True,p
mlb_df['tb'] = mlb_df['tb'].apply(lambda x: np.random.choice(options7,1, True,per
mlb_df['gidp'] = mlb_df['gidp'].apply(lambda x: np.random.choice(options8,1, True
```

option 2

In [9]:
```
#[mlb_df[col].fillna(mlb_df[col].mean(), inplace=True) for col in mlb_df.columns]
```

In [10]:
```
mlb_df.inducted_y.fillna(0, inplace=True)
```

In [11]:
```
mlb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2316 entries, aaronha01 to zuninmi01
Data columns (total 37 columns):
g                            2316 non-null int64
ab                           2316 non-null int64
r                            2316 non-null int64
h                            2316 non-null int64
2b                           2316 non-null int64
3b                           2316 non-null int64
hr                           2316 non-null int64
rbi                          2316 non-null float64
sb                           2316 non-null float64
bb                           2316 non-null int64
so                           2316 non-null float64
asg_mvp                      2316 non-null float64
baberuth_award               2316 non-null float64
baseball_magazine_allstar    2316 non-null float64
comeback_poy                 2316 non-null float64
gold_glove_award             2316 non-null float64
```

In [12]:
```
mlb_df.inducted_y.value_counts()
```

Out[12]:
```
0.000    2171
1.000     145
Name: inducted_y, dtype: int64
```

In [13]:
```
# save df at this point to use for models
mlb_df.to_pickle('final_df1.pkl')
```

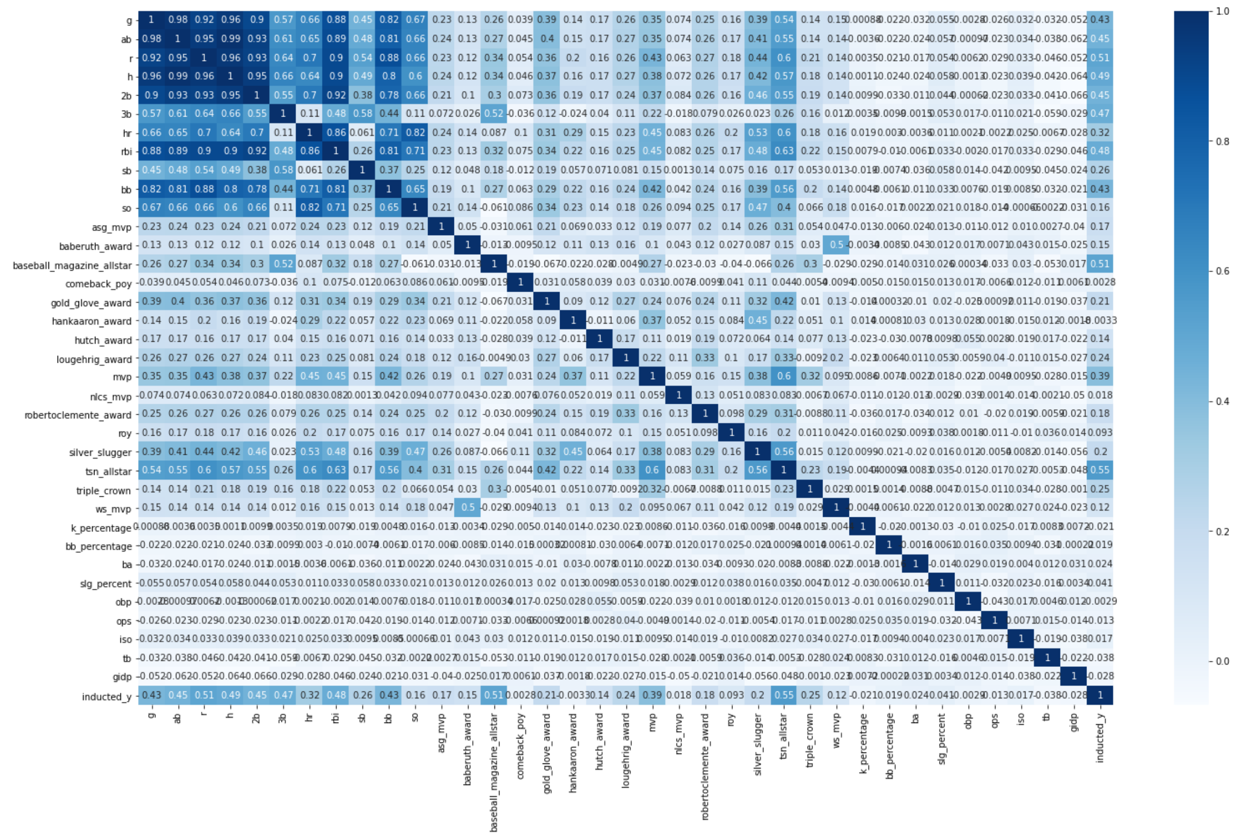
In [14]:
```
corr = mlb_df.corr()
corr
```

Out[14]:

	g	ab	r	h	2b	3b	hr	rbi	sb	
g	1.000	0.978	0.923	0.957	0.904	0.566	0.656	0.876	0.447	0.8
ab	0.978	1.000	0.953	0.988	0.934	0.613	0.653	0.891	0.485	0.8
r	0.923	0.953	1.000	0.964	0.928	0.641	0.705	0.900	0.536	0.8
h	0.957	0.988	0.964	1.000	0.949	0.657	0.637	0.900	0.488	0.7
2b	0.904	0.934	0.928	0.949	1.000	0.546	0.702	0.918	0.380	0.7
3b	0.566	0.613	0.641	0.657	0.546	1.000	0.106	0.479	0.580	0.4
hr	0.656	0.653	0.705	0.637	0.702	0.106	1.000	0.863	0.061	0.7
rbi	0.876	0.891	0.900	0.900	0.918	0.479	0.863	1.000	0.259	0.8
sb	0.447	0.485	0.536	0.488	0.380	0.580	0.061	0.259	1.000	0.3
bb	0.823	0.807	0.878	0.799	0.782	0.440	0.708	0.812	0.373	1.0
so	0.668	0.656	0.655	0.598	0.656	0.108	0.823	0.709	0.249	0.0

In [15]:
```
fig, ax = plt.subplots(figsize=(24,14))
sns.heatmap(corr, annot=True, cmap='Blues')
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x18f82229908>
```



Model #1: Vanilla Model

starting off with a decision tree as my baseline model because as you can see from above, I have multicolinearity between a few of my predictors so using a Decision Tree is best because multicolinearity does not have an affect on this type of model.

In [16]:
```
y = mlb_df['inducted_y']
X = mlb_df.drop(columns='inducted_y')

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, stratify=
```

In [17]:
```
# scale data
ss = StandardScaler()

# instantiate and fit
ss_X_train = ss.fit_transform(X_train)
ss_X_test = ss.transform(X_test)
```

In [18]:
```
# scale data
mm = MinMaxScaler()

# instantiate and fit
mm_X_train = mm.fit_transform(X_train)
mm_X_test = mm.transform(X_test)
```

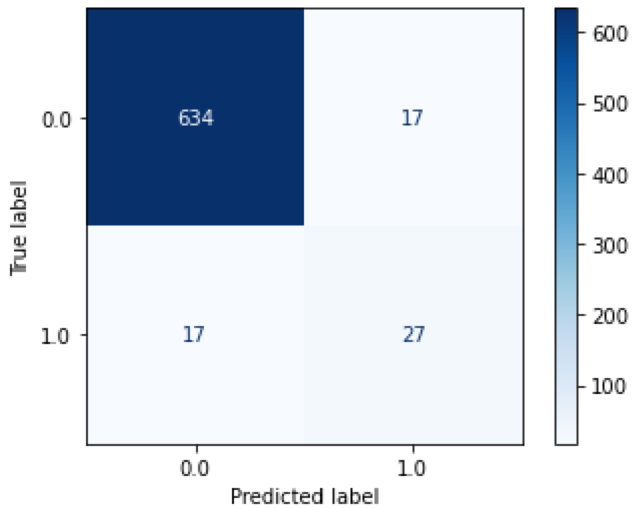
In [19]:
```
# instantiate and fit
dt_clf = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)
dt_clf.fit(mm_X_train, y_train)
```

Out[19]:
```
DecisionTreeClassifier(max_depth=5, random_state=42)
```

In [20]:
```
# run prediction on min-max scaled test data -- scored significantly higher than :
test_preds = dt_clf.predict(mm_X_test)

# confusion matrix and classification report
def print_metrics(labels, preds):
    print(confusion_matrix(labels, preds))
    print(classification_report(labels, preds))
    print('Accuracy score: ',accuracy_score(labels, preds))
    print('Recall score: ',recall_score(labels, preds))
    print('Precision score: ',precision_score(labels, preds))
    print('F1 score: ',f1_score(labels, preds))

print_metrics(y_test, test_preds)
```

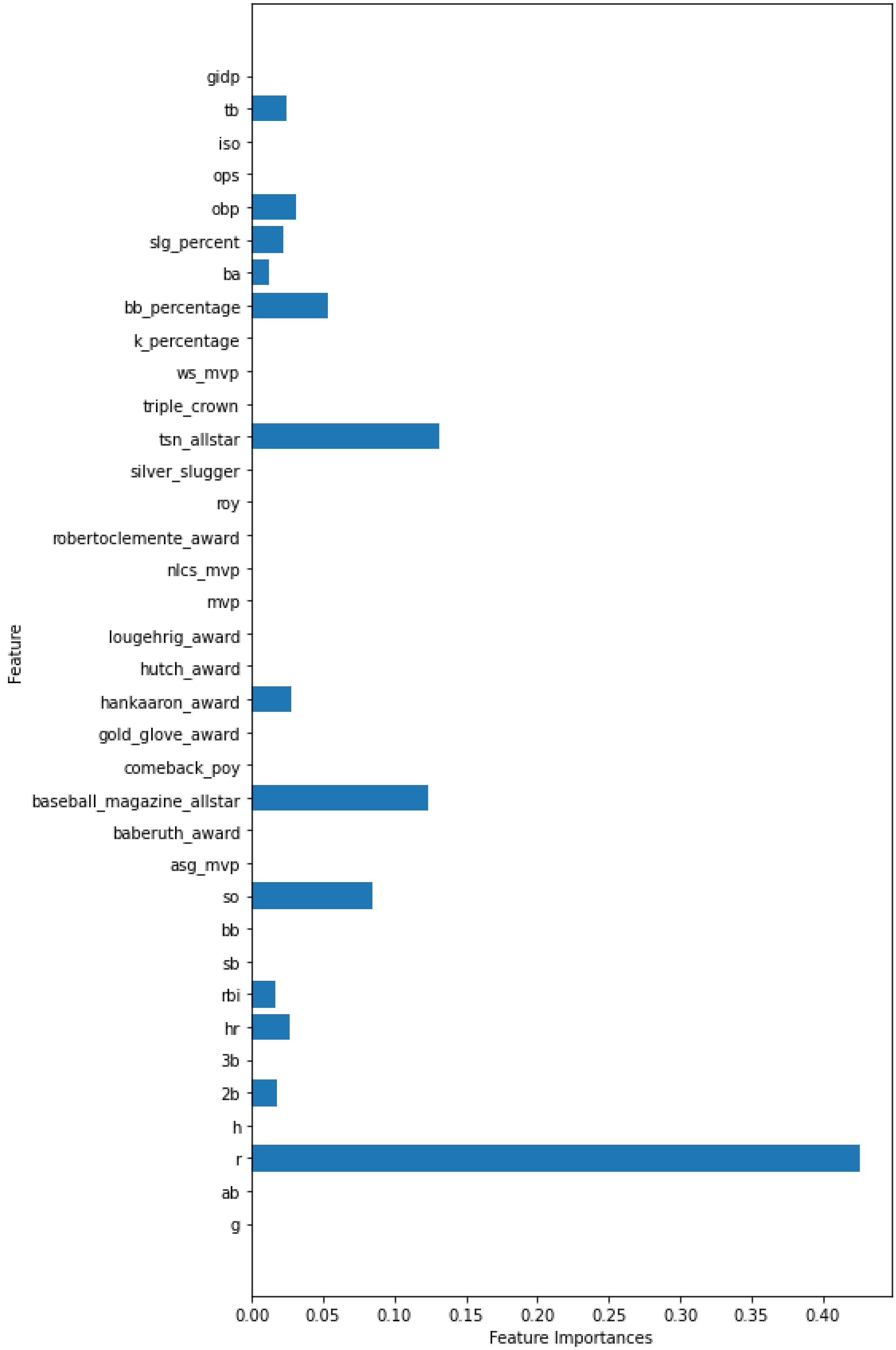
```
[[634  17]
 [ 17  27]]
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       651
         1.0       0.61      0.61      0.61        44

    accuracy                           0.95       695
   macro avg       0.79      0.79      0.79       695
weighted avg       0.95      0.95      0.95       695

Accuracy score:  0.9510791366906475
Recall score:  0.6136363636363636
Precision score:  0.6136363636363636
F1 score:  0.6136363636363636
```

In [21]:
```
# plotting confusion matrix
plot_confusion_matrix(dt_clf, mm_X_test, y_test, cmap="Blues")
plt.show()
```



- 96% accuracy is good, but still 18 FN is pretty high when considering there were only 44 HOFers in this test data
- Accuracy is also susceptible to a false high accuracy if there is high class imbalance, which I have. After further evaluation I will try running model using SMOTE, which creates artificial data for the minority class, which will rid the imbalance

In [22]:
```
# plotting feature importances
def plot_feature_importances(model):
    n_features = mm_X_train.shape[1]
    plt.figure(figsize=(8,12))
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), X_train.columns.values)
    plt.xlabel('Feature Importances')
    plt.ylabel('Feature')

plot_feature_importances(dt_clf)
plt.tight_layout()
plt.savefig('./images/feature_importances_vanilla.png')
```



As you can see, the model values being named to the All-Star team as the most important predictor. This is not surprising because you become a HOF player by dominating over many years of your playing career. If you are doing so, you are likely to be named to the All-Star team more than the average player on a year-by-year basis.

What is surprising is that I would have expected Hits and HRs to be of more importance because looking at the top 20 players in each category, the majority of them are HOFers.

In [23]:
```
# false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
# roc_auc = auc(false_positive_rate, true_positive_rate)
# print('Accuracy is: {0}'.format(acc))
```

Model #2: SMOTE

Not only do I have class imbalance, but also have a limited number of positives in the target class. Using SMOTE, we can artifically create more training data to better balance the data. Let's see how this affects our model

In [24]:
```
# positives before SMOTE
y_train.value_counts()
```

Out[24]:
```
0.000    1520
1.000     101
Name: inducted_y, dtype: int64
```

In [25]:
```
# positives after resampling with SMOTE
X_train_resampled, y_train_resampled = SMOTE().fit_resample(X_train, y_train)
print(pd.Series(y_train_resampled).value_counts())
```

```
1.000    1520
0.000    1520
Name: inducted_y, dtype: int64
```

In [26]:
```
# run another DT on resampled training data - artificially created more HOFers us
clf_dt2 = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)

# fit the model
clf_dt2.fit(X_train_resampled, y_train_resampled)

# prediction for training data
train_pred_smote = clf_dt2.predict(X_train_resampled)

# print metrics
print_metrics(y_train_resampled, train_pred_smote)
```

```
[[1445   75]
 [  13 1507]]
              precision    recall  f1-score   support

         0.0       0.99      0.95      0.97      1520
         1.0       0.95      0.99      0.97      1520

    accuracy                           0.97      3040
   macro avg       0.97      0.97      0.97      3040
weighted avg       0.97      0.97      0.97      3040

Accuracy score:  0.9710526315789474
Recall score:  0.9914473684210526
Precision score:  0.9525916561314791
F1 score:  0.9716312056737589
```

Training data is acting as a pretty effective training set after incorporating SMOTE. Let's see how the model now performs on the test data.

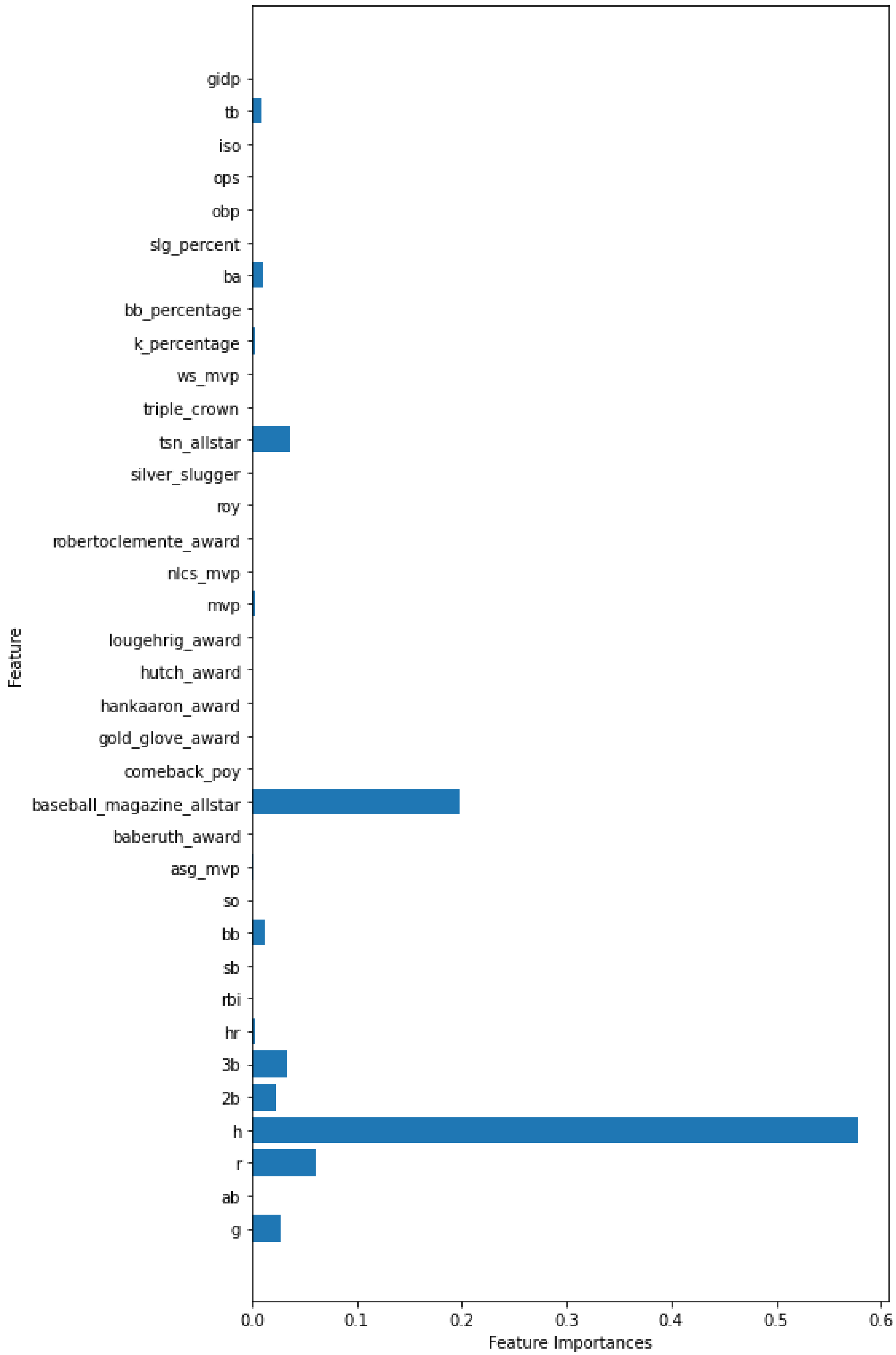
In [27]:
```
# prediction for testing data
test_pred = clf_dt2.predict(X_test)

# print metrics
print_metrics(y_test, test_pred)
```

```
[[612  39]
 [ 10  34]]
              precision    recall  f1-score   support

         0.0       0.98      0.94      0.96       651
         1.0       0.47      0.77      0.58        44

    accuracy                           0.93       695
   macro avg       0.72      0.86      0.77       695
weighted avg       0.95      0.93      0.94       695

Accuracy score:  0.9294964028776979
Recall score:  0.7727272727272727
Precision score:  0.4657534246575342
F1 score:  0.5811965811965812
```

- Recall increased significantly from 60% to now 77%.
- Precision decreased from 65% to 47%.

- Accuracy and F1 decreased slightly.

In [28]:
```
plot_feature_importances(clf_dt2)
plt.tight_layout()
plt.savefig('./images/feature_importances_best.png')
```



After 2nd round of EDA, I decided to remove the following non-HOFers due to them being banned by the MLB for various reasons i.e. steroid use. This should reduce confusion for the model as these players have some of the best stats in the history of the game, but are not in the HOF.

In [29]:
```
mlb_df.inducted_y.value_counts()
```

Out[29]:
```
0.000    2171
1.000     145
Name: inducted_y, dtype: int64
```

In [30]:
```
mlb_df.drop(['rosepe01','rodrial01','bondsba01','sosasa01','mcgwima01','ramirma02
             'palmera01','ortizda01'], axis=0, inplace=True)
```

In [31]:
```
mlb_df.inducted_y.value_counts()
```

Out[31]:
```
0.000    2163
1.000     145
Name: inducted_y, dtype: int64
```

In [32]:
```
mlb_df.to_pickle('final_df2_removed_banned_players.pkl')
```

Model #3: DT w/ Refined DB and Balanced class_weight

In [33]:
```
mlb_df = pd.read_pickle('final_df2_removed_banned_players.pkl')
mlb_df.head()
```

Out[33]:

	g	ab	r	h	2b	3b	hr	rbi	sb	bb	...	k_percentage	bb
playerID													
aaronha01	3298	12364	2174	3771	624	98	755	2297.000	240.000	1402	...	16.200	
abbated01	827	2942	346	748	95	43	11	310.000	138.000	281	...	15.767	
abbotku01	702	2044	273	523	109	23	62	242.000	22.000	133	...	10.900	
abreubo01	2425	8480	1453	2470	574	59	288	1363.000	400.000	1476	...	22.020	
abreujo02	901	3547	483	1038	218	14	179	611.000	10.000	245	...	6.800	

5 rows × 37 columns

In [34]:
```
mlb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2308 entries, aaronha01 to zuninmi01
Data columns (total 37 columns):
g                            2308 non-null int64
ab                           2308 non-null int64
r                            2308 non-null int64
h                            2308 non-null int64
2b                           2308 non-null int64
3b                           2308 non-null int64
hr                           2308 non-null int64
rbi                          2308 non-null float64
sb                           2308 non-null float64
bb                           2308 non-null int64
so                           2308 non-null float64
asg_mvp                      2308 non-null float64
baberuth_award               2308 non-null float64
baseball_magazine_allstar    2308 non-null float64
comeback_poy                 2308 non-null float64
gold_glove_award             2308 non-null float64
```

In [35]:
```
mlb_df.inducted_y.value_counts()
```

Out[35]:
```
0.000    2163
1.000     145
Name: inducted_y, dtype: int64
```

In [36]:
```
# need to re-assign the train/test after removing banned players above -- counts 
y = mlb_df['inducted_y']
X = mlb_df.drop(columns='inducted_y')

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, stratify=
```

In [37]:
```
# instantiate SMOTE
X_train_resampled, y_train_resampled = SMOTE().fit_resample(X_train, y_train)
print(pd.Series(y_train_resampled).value_counts())
```

```
0.000    1514
1.000    1514
Name: inducted_y, dtype: int64
```

In [38]:
```
# instantiate and fit
dt_clf3 = DecisionTreeClassifier(criterion='gini', max_depth=5, class_weight='bal
dt_clf3.fit(X_train, y_train)
```

Out[38]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=5, random_state=42)
```

In [39]:
```
# predict on test data
test_preds = dt_clf3.predict(X_test)


# print metrics on test data
print_metrics(y_test, test_preds)
```

```
[[613  36]
 [ 10  34]]
              precision    recall  f1-score   support

         0.0       0.98      0.94      0.96       649
         1.0       0.49      0.77      0.60        44

    accuracy                           0.93       693
   macro avg       0.73      0.86      0.78       693
weighted avg       0.95      0.93      0.94       693

Accuracy score:  0.9336219336219336
Recall score:  0.7727272727272727
Precision score:  0.4857142857142857
F1 score:  0.5964912280701754
```

Accuracy and recall actually decreased some, while Precision and F1 increased

Evaluation:

Prioritizing recall was my main effort here due to the fact that we want to limit our FNs, which in this case would be HOF deserving players not being inducted into the HOF.

Conclusion:

In conclusion, I have found the most important features and classified a HOF player in my best model at a 86% recall and 94% accuracy. This is a fairly effective model, as it does accurately predict 94% of the data and only misses 14% of TPs, but it does not meet the requirement of within 5% to reject the null hypothesis.

Recommendations:

The most important stats when evaluating a HOF player is Hits, Runs, and All-Star games made. With this newfound information, I recommend to my client that they should use these metrics when negotiating their current contracts as well as look for active and upcoming players with these stats in mind. They will pay off large when these players sign their hundred-million dollar contracts!

Future Work:

While I am happy about how well my model performed with the data provided, I would love to dig deeper into sabermetrics and work with more advanced baseball stats like OPs+, WOBA, and WAR.

The issue with calculating these stats is that they need to account for the time periods in which the players played, and with a short period of time to complete this assignment, it was not enough time to account for all of that.