In [95]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import auc
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import roc_curve
```

In [2]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.svm import SVC
from catboost import CatBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from mlxtend.classifier import EnsembleVoteClassifier
```

In [3]:
```
#validation & tuning
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

from sklearn.preprocessing import StandardScaler
```

In [4]:
```
print(os.getcwd())
os.chdir('D:/OneDrive/BLOGS/League of Legends')
print(os.getcwd())
```

```
C:\Users\Jinhang Jiang
D:\OneDrive\BLOGS\League of Legends
```

In [373]:
```
info = pd.read_csv("high_diamond_ranked_10min.csv")
```

In [295]:
```
data = pd.read_csv("high_diamond_ranked_10min.csv")
```

In [372]:
```
info
```

Out[372]:

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	bluel
0	4519157822	0	28	2	1	9	
1	4523371949	0	12	1	0	5	
2	4521474530	0	15	0	0	7	
3	4524384067	0	43	1	0	4	
4	4436033771	0	75	4	0	6	
...	...	...	...	...	...	...	
9874	4527873286	1	17	2	1	7	
9875	4527797466	1	54	0	0	6	
9876	4527713716	0	23	1	0	6	
9877	4527628313	0	14	4	1	2	
9878	4523772935	1	18	0	1	6	

9879 rows × 46 columns

In [297]:
```
info.columns
```

Out[297]:
```
Index(['gameId', 'blueWins', 'blueWardsPlaced', 'blueWardsDestroyed',
       'blueFirstBlood', 'blueKills', 'blueDeaths', 'blueAssists',
       'blueEliteMonsters', 'blueDragons', 'blueHeralds',
       'blueTowersDestroyed', 'blueTotalGold', 'blueAvgLevel',
       'blueTotalExperience', 'blueTotalMinionsKilled',
       'blueTotalJungleMinionsKilled', 'blueGoldDiff', 'blueExperienceDiff',
       'blueCSPerMin', 'blueGoldPerMin', 'redWardsPlaced', 'redWardsDestroyed',
       'redFirstBlood', 'redKills', 'redDeaths', 'redAssists',
       'redEliteMonsters', 'redDragons', 'redHeralds', 'redTowersDestroyed',
       'redTotalGold', 'redAvgLevel', 'redTotalExperience',
       'redTotalMinionsKilled', 'redTotalJungleMinionsKilled', 'redGoldDiff',
       'redExperienceDiff', 'redCSPerMin', 'redGoldPerMin'],
      dtype='object')
```

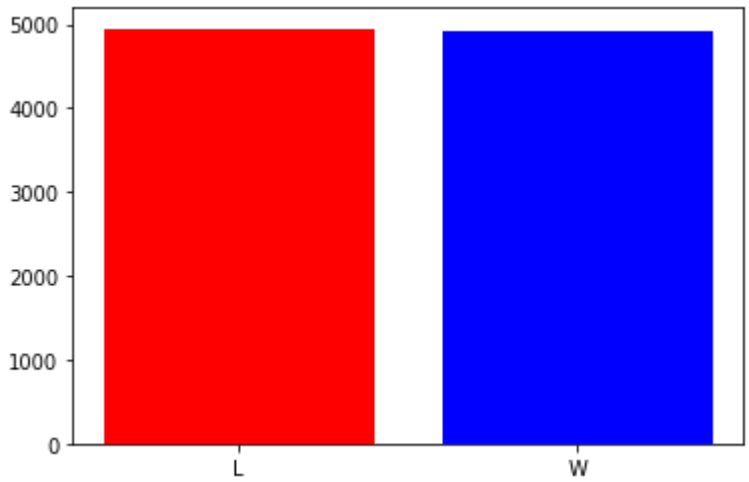
In [426]:
```
info.blueWins.value_counts(normalize=True)
```

Out[426]:
```
0    0.500962
1    0.499038
Name: blueWins, dtype: float64
```

Plots

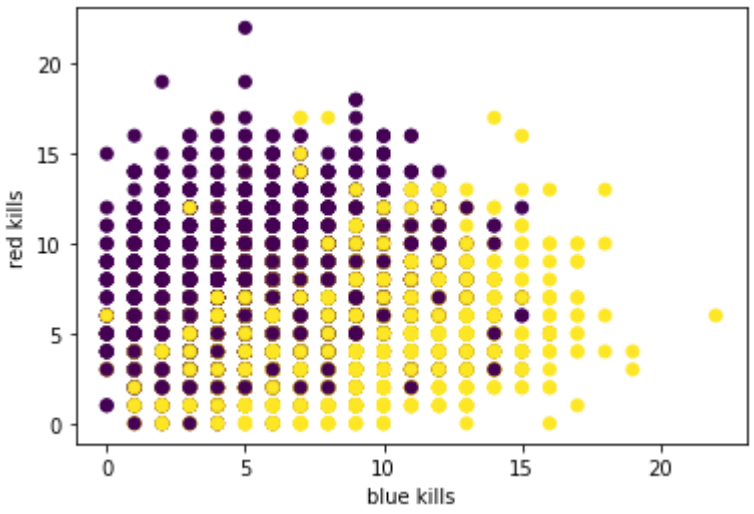
In [299]:
```
plt.bar(info.blueWins.unique(),info.blueWins.value_counts(),color=("r","b"),tick_label=("L
```

Out[299]: <BarContainer object of 2 artists>



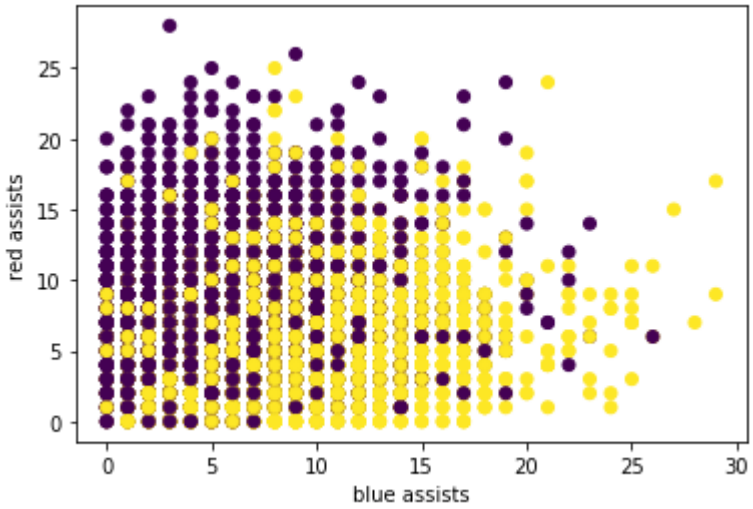
In [427]:
```
plt.scatter(info.blueKills,info.redKills,c=info.blueWins)
plt.xlabel("blue kills")
plt.ylabel("red kills")
```

Out[427]: Text(0, 0.5, 'red kills')



In [428]:
```
plt.scatter(info.blueAssists,info.redAssists,c=info.blueWins)
plt.xlabel("blue assists")
plt.ylabel("red assists")
```

Out[428]: Text(0, 0.5, 'red assists')

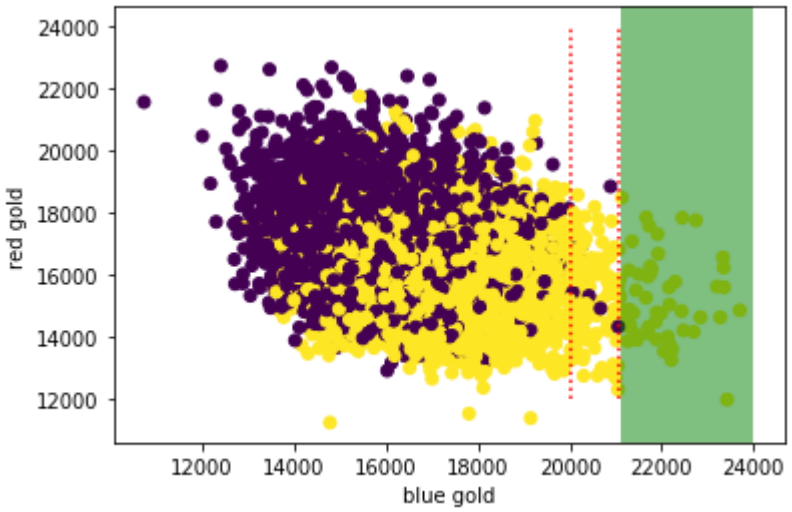


In [302]:
```
info.blueTotalGold.where(info.blueWins==0).sort_values(ascending=False).head(2)
```

Out[302]:
```
9608    21055.0
6137    20887.0
Name: blueTotalGold, dtype: float64
```

In [430]:
```
plt.scatter(data.blueTotalGold,data.redTotalGold,c=data.blueWins)
plt.vlines(21056,12000,24000,color= "red", linestyle=':')
plt.vlines(20000,12000,24000,color= "red", linestyle=':')
plt.axvspan(21100, 24000, color='g', alpha=0.5, lw=0)
plt.xlabel("blue gold")
plt.ylabel("red gold")
```

Out[430]: Text(0, 0.5, 'red gold')

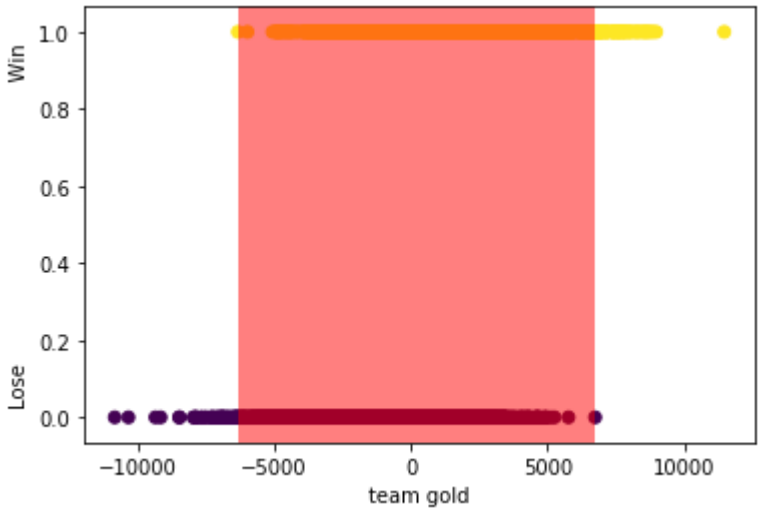


In [304]:
```
print(info.blueGoldDiff.where(info.blueWins==0).sort_values(ascending=False).head(1))
print(info.blueGoldDiff.where(info.blueWins==1).sort_values(ascending=True).head(1))
```

```
9608    6744.0
Name: blueGoldDiff, dtype: float64
8459   -6324.0
Name: blueGoldDiff, dtype: float64
```

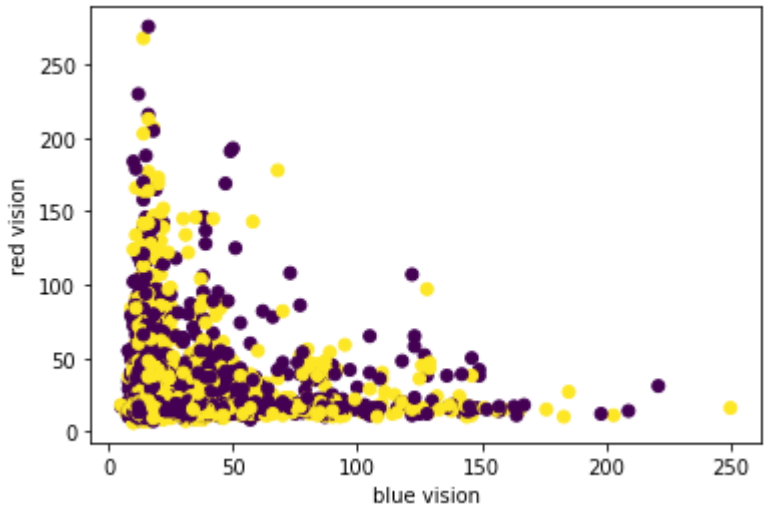
In [434]:
```
plt.scatter(data.blueGoldDiff, data.blueWins, c=data.blueWins)
plt.axvspan(-6324.0, 6744.0, color='r', alpha=0.5, lw=0)
plt.ylabel("Lose                                                    Win")
plt.xlabel("team gold")
```

Out[434]: Text(0.5, 0, 'team gold')



In [439]:
```
plt.scatter(info.blueWardsPlaced, info.redWardsPlaced, c=info.blueWins)
plt.xlabel("blue vision")
plt.ylabel("red vision")
```

Out[439]: Text(0, 0.5, 'red vision')



In [307]:
```
info.columns
```

Out[307]:
```
Index(['gameId', 'blueWins', 'blueWardsPlaced', 'blueWardsDestroyed',
       'blueFirstBlood', 'blueKills', 'blueDeaths', 'blueAssists',
       'blueEliteMonsters', 'blueDragons', 'blueHeralds',
       'blueTowersDestroyed', 'blueTotalGold', 'blueAvgLevel',
       'blueTotalExperience', 'blueTotalMinionsKilled',
       'blueTotalJungleMinionsKilled', 'blueGoldDiff', 'blueExperienceDiff',
       'blueCSPerMin', 'blueGoldPerMin', 'redWardsPlaced', 'redWardsDestroyed',
       'redFirstBlood', 'redKills', 'redDeaths', 'redAssists',
       'redEliteMonsters', 'redDragons', 'redHeralds', 'redTowersDestroyed',
       'redTotalGold', 'redAvgLevel', 'redTotalExperience',
       'redTotalMinionsKilled', 'redTotalJungleMinionsKilled', 'redGoldDiff',
       'redExperienceDiff', 'redCSPerMin', 'redGoldPerMin'],
      dtype='object')
```

In [308]:
```
corr_matrix=info.drop(["gameId",'redWardsPlaced', 'redWardsDestroyed',
       'redFirstBlood', 'redKills', 'redDeaths', 'redAssists',
       'redEliteMonsters', 'redDragons', 'redHeralds', 'redTowersDestroyed',
       'redTotalGold', 'redAvgLevel', 'redTotalExperience',
       'redTotalMinionsKilled', 'redTotalJungleMinionsKilled', 'redGoldDiff',
       'redExperienceDiff', 'redCSPerMin', 'redGoldPerMin'],axis=1).corr(method='pearson')
corr_matrix.iloc[:,0]
```

Out[308]:
```
blueWins                      1.000000
blueWardsPlaced               0.000087
blueWardsDestroyed            0.044247
blueFirstBlood                0.201769
blueKills                     0.337358
blueDeaths                   -0.339297
blueAssists                   0.276685
blueEliteMonsters             0.221944
blueDragons                   0.213768
blueHeralds                   0.092385
blueTowersDestroyed           0.115566
blueTotalGold                 0.417213
blueAvgLevel                  0.357820
blueTotalExperience           0.396141
blueTotalMinionsKilled        0.224909
blueTotalJungleMinionsKilled  0.131445
blueGoldDiff                  0.511119
blueExperienceDiff            0.489558
blueCSPerMin                  0.224909
blueGoldPerMin                0.417213
Name: blueWins, dtype: float64
```

In [309]:
```
data.drop(["gameId",'redWardsPlaced', 'redWardsDestroyed',
       'redFirstBlood', 'redKills', 'redDeaths', 'redAssists',
       'redEliteMonsters', 'redDragons', 'redHeralds', 'redTowersDestroyed',
       'redTotalGold', 'redAvgLevel', 'redTotalExperience',
       'redTotalMinionsKilled', 'redTotalJungleMinionsKilled', 'redGoldDiff',
       'redExperienceDiff', 'redCSPerMin', 'redGoldPerMin'],axis=1).describe()
```

Out[309]:

	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	blueDeaths
count	9879.000000	9879.000000	9879.000000	9879.000000	9879.000000	9879.00000
mean	0.499038	22.288288	2.824881	0.504808	6.183925	6.13766
std	0.500024	18.019177	2.174998	0.500002	3.011028	2.93381
min	0.000000	5.000000	0.000000	0.000000	0.000000	0.00000
25%	0.000000	14.000000	1.000000	0.000000	4.000000	4.00000
50%	0.000000	16.000000	3.000000	1.000000	6.000000	6.00000
75%	1.000000	20.000000	4.000000	1.000000	8.000000	8.00000
max	1.000000	250.000000	27.000000	1.000000	22.000000	22.00000

In [310]:
```
data.blueWins.where(((data.blueKills+data.blueAssists)/data.blueDeaths)>=3).dropna().value
```

Out[310]:
```
1.0    0.765084
0.0    0.234916
Name: blueWins, dtype: float64
```

In [311]:
```
data.blueWins.where(data.blueTowersDestroyed>=1).dropna().value_counts(normalize=True)
```

Out[311]:
```
1.0    0.75431
0.0    0.24569
Name: blueWins, dtype: float64
```

In [312]:
```
data.blueWins.where(data.blueKills>=8).dropna().value_counts(normalize=True)
```

Out[312]:
```
1.0    0.69858
0.0    0.30142
Name: blueWins, dtype: float64
```

Feature Engineering

In [388]:
```
info["blueKDA"]=3
for i in range(0,len(info)):
    if data.blueDeaths[i] != 0:
        info["blueKDA"][i]=((data.blueKills[i]+data.blueAssists[i])/data.blueDeaths[i]).ro
    else:
        0
```

In [390]:
```
info["redKDA"]=3
for i in range(0,len(info)):
    if data.redDeaths[i]!=0:
        info["redKDA"][i] = ((data.redKills[i]+data.redAssists[i])/data.redDeaths[i]).roun
    else:
        0
```

In [391]:
```
info["blueKDADiff"]=info["blueKDA"]-info["redKDA"]
```

In [378]:
```
info["blueGoldAdv"]=3
for i in range(0,len(info)):
    if info["blueTotalGold"][i] >= info["blueTotalGold"][3464]:
        info["blueGoldAdv"][i] = 1
    else:
        info["blueGoldAdv"][i] = 0
```

In [379]:
```
info["blueDiffNeg"]=3
for i in range(0,len(info)):
    if info["blueGoldDiff"][i] <= info["blueGoldDiff"][8459]:
        info["blueDiffNeg"][i] = 1
    else:
        info["blueDiffNeg"][i] = 0
```

In [380]:
```
info["blueDiffPos"]=3
for i in range(0,9879):
    if info["blueGoldDiff"][i] >= info["blueGoldDiff"][9608]:
        info["blueDiffPos"][i] = 1
    else:
        info["blueDiffPos"][i] = 0
```

In [326]:
```
cols=['gameId', 'blueWins', 'blueWardsPlaced', 'blueWardsDestroyed',
       'blueFirstBlood', 'blueKills', 'blueDeaths', 'blueAssists',
       'blueEliteMonsters', 'blueDragons', 'blueHeralds',
       'blueTowersDestroyed', 'blueTotalGold', 'blueAvgLevel',
       'blueTotalExperience', 'blueTotalMinionsKilled',
       'blueTotalJungleMinionsKilled', 'blueGoldDiff', 'blueExperienceDiff',
       'blueCSPerMin', 'blueGoldPerMin']
info[cols]
```

Out[326]:

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	bluel
0	4519157822	0	28	2	1	9	
1	4523371949	0	12	1	0	5	
2	4521474530	0	15	0	0	7	
3	4524384067	0	43	1	0	4	
4	4436033771	0	75	4	0	6	
...	...	...	...	...	...	...	
9874	4527873286	1	17	2	1	7	
9875	4527797466	1	54	0	0	6	
9876	4527713716	0	23	1	0	6	
9877	4527628313	0	14	4	1	2	
9878	4523772935	1	18	0	1	6	

9879 rows × 21 columns

In [392]:
```
cols = ['blueTotalGold','blueTotalExperience','blueTotalMinionsKilled',
        'blueGoldDiff','blueExperienceDiff','blueGoldPerMin',
        'redTotalGold','redTotalExperience','redTotalMinionsKilled',
        'redGoldDiff','redExperienceDiff','redGoldPerMin']
standard = StandardScaler()
info[cols]=standard.fit_transform(info[cols])
```

In [393]:
```
info_x=info.drop(["gameId","blueWins"],axis=1)
info_y=info.blueWins
```

In [329]:
```
#info_x.to_csv("X.csv",index=False)
#info_y.to_csv("Y.csv",index=False)
```

In [423]:
```
cv=KFold(n_splits = 5, random_state=2022,shuffle=True)
```

In [702]:
```
#train, holdout = train_test_split(info,random_state = 2020, test_size = 0.3)
```

In [703]:
```
#train.blueWins.value_counts()
```

Out[703]:
```
0    3509
1    3406
Name: blueWins, dtype: int64
```

In [704]:
```
#holdout.blueWins.value_counts()
```

Out[704]:
```
1    1524
0    1440
Name: blueWins, dtype: int64
```

In [705]:
```
#train.to_csv("train.csv",index=False)
#holdout.to_csv("holdout.csv",index=False)
```

In [706]:
```
#y_train = train.blueWins
#X_train = train.drop(["gameId","blueWins"],axis=1)
```

In [707]:
```
#y_test = holdout.blueWins
#X_test = holdout.drop(["gameId","blueWins"],axis=1)
```

KNN

In [246]:
```
np.sqrt(len(info_x))
```

Out[246]:
```
99.39315871829409
```

In [248]:
```
knn_params = {"n_neighbors" :np.arange(95,105),
              "weights" : ["distance"],
              "algorithm" : ["ball_tree"] , #,"kd_tree","brute"
              "leaf_size":[1,2]}
```

In [394]:
```
knn = KNeighborsClassifier()
grid_knn = GridSearchCV(knn, knn_params,cv=5,verbose=2,n_jobs=-1)
grid_knn.fit(info_x,info_y)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   18.5s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   53.7s finished
```

Out[394]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'algorithm': ['ball_tree'], 'leaf_size': [1, 2],
                         'n_neighbors': array([ 95,  96,  97,  98,  99, 100, 101, 102,
103, 104]),
                         'weights': ['distance']},
             verbose=2)
```

In [395]:
```
knn_best_params=grid_knn.best_params_
knn_best_params
```

Out[395]:
```
{'algorithm': 'ball_tree',
 'leaf_size': 1,
 'n_neighbors': 98,
 'weights': 'distance'}
```

In [396]:
```
knn = KNeighborsClassifier(algorithm="ball_tree",n_neighbors=98,weights="distance",leaf_si
```

In [397]:
```
np.mean(cross_val_score(knn,info_x,info_y,cv=cv,scoring="accuracy",n_jobs=-1))
```

Out[397]:
```
0.7061440065597294
```

Logistic Regression

In [50]:
```
log_params = {"penalty":["l1","l2","elasticnet"],
              "solver":['newton-cg', 'lbfgs', 'sag', 'saga'],
              "C":[1,5,10,20,30,50,100],
              "warm_start":[True,False]}
```

In [51]:
```
logreg=LogisticRegression()
```

In [52]:
```
log_grid = GridSearchCV(logreg,log_params,cv=cv,verbose=2,n_jobs=-1)
```

In [398]:
```
log_grid.fit(info_x,info_y)
```

```
Fitting 5 folds for each of 168 candidates, totalling 840 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done 124 tasks      | elapsed:    5.9s
[Parallel(n_jobs=-1)]: Done 840 out of 840 | elapsed:   40.8s finished
```

Out[398]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=2020, shuffle=False),
             estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [1, 5, 10, 20, 30, 50, 100],
                         'penalty': ['l1', 'l2', 'elasticnet'],
                         'solver': ['newton-cg', 'lbfgs', 'sag', 'saga'],
                         'warm_start': [True, False]},
             verbose=2)
```

In [399]:
```
log_grid.best_params_
```

Out[399]:
```
{'C': 50, 'penalty': 'l2', 'solver': 'lbfgs', 'warm_start': True}
```

In [409]:
```
logreg=LogisticRegression(penalty="l2",solver="sag",C=50, warm_start=False)
```

CatBoostClassifier

In [400]:
```
cat_params = {"learning_rate":[0.005],
              "depth":[6,8]}
```

In [404]:
```
cat = CatBoostClassifier()
```

In [405]:
```
cat_grid = GridSearchCV(cat,cat_params,cv=cv)
```

In [406]:
```
cat_grid.fit(info_x,info_y)
```

```
985:    learn: 0.4826369        total: 14.2s    remaining: 202ms
986:    learn: 0.4826135        total: 14.2s    remaining: 188ms
987:    learn: 0.4825664        total: 14.3s    remaining: 173ms
988:    learn: 0.4825577        total: 14.3s    remaining: 159ms
989:    learn: 0.4825055        total: 14.3s    remaining: 144ms
990:    learn: 0.4824514        total: 14.3s    remaining: 130ms
991:    learn: 0.4823349        total: 14.3s    remaining: 115ms
992:    learn: 0.4823194        total: 14.3s    remaining: 101ms
993:    learn: 0.4822731        total: 14.3s    remaining: 86.5ms
994:    learn: 0.4821879        total: 14.4s    remaining: 72.1ms
995:    learn: 0.4821510        total: 14.4s    remaining: 57.7ms
996:    learn: 0.4821023        total: 14.4s    remaining: 43.3ms
997:    learn: 0.4820698        total: 14.4s    remaining: 28.8ms
998:    learn: 0.4819986        total: 14.4s    remaining: 14.4ms
999:    learn: 0.4819533        total: 14.4s    remaining: 0us
```

Out[406]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=2020, shuffle=False),
             estimator=<catboost.core.CatBoostClassifier object at 0x0000020F2F2E01C8
>,
             param_grid={'depth': [6, 8], 'learning_rate': [0.005]})
```

In [407]:
```
cat_grid.best_params_
```

Out[407]:
```
{'depth': 8, 'learning_rate': 0.005}
```

In [408]:
```
cat = CatBoostClassifier(learning_rate=0.005,depth=8,verbose=False)
```

SVC

In [412]:
```
svc_params={'kernel':['poly','sigmoid'],
            'C':[1,10,20,50,100]}
```

In [413]:
```
svc=SVC()
```

In [414]:
```
svc_grid=GridSearchCV(svc,svc_params,cv=cv,verbose=2,n_jobs=-1)
```

In [415]:
```
svc_grid.fit(info_x,info_y)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   47.6s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  1.6min finished
```

Out[415]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=2020, shuffle=False),
             estimator=SVC(), n_jobs=-1,
             param_grid={'C': [1, 10, 20, 50, 100],
                         'kernel': ['poly', 'sigmoid']},
             verbose=2)
```

In [416]:
```
svc_grid.best_params_
```

Out[416]:
```
{'C': 20, 'kernel': 'poly'}
```

In [444]:
```
svc=SVC(kernel="poly",probability=True,C=20)
```

In [90]:
```
np.mean(cross_val_score(svc,info_x,info_y,cv=cv,scoring="accuracy",n_jobs=-1))
```

Out[90]:
```
0.728008404653308
```

AdaClassifier

In [56]:
```
ada_params={"learning_rate":[0.05,0.1,0.2,0.5,1],
            "algorithm":['SAMME', 'SAMME.R'],
            "n_estimators":[50,100,150,500]}
```

In [57]:
```
ada = AdaBoostClassifier()
```

In [58]:
```
ada_grid = GridSearchCV(ada,ada_params,cv=cv,verbose=2)
```

In [59]:
```
ada_grid.fit(info_x,info_y)
```

```
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=100, total=   1.4s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=100 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=100, total=   1.4s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=150 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=150, total=   2.1s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=150 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=150, total=   2.1s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=150 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=150, total=   2.1s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=150 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=150, total=   2.1s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=150 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=150, total=   2.1s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=500 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=500, total=   7.2s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=500 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=500, total=   7.1s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=500 ...........
[CV]  algorithm=SAMME, learning_rate=0.1, n_estimators=500, total=   7.0s
[CV] algorithm=SAMME, learning_rate=0.1, n_estimators=500
```

In [60]:
```
ada_grid.best_params_
```

Out[60]:
```
{'algorithm': 'SAMME.R', 'learning_rate': 0.05, 'n_estimators': 150}
```

In [445]:
```
ada_best_params = {"learning_rate":0.05,
                   "algorithm":"SAMME.R",
                   "n_estimators":150}
```

In [446]:
```
ada=AdaBoostClassifier(**ada_best_params)
```

In [67]:
```
np.mean(cross_val_score(ada,info_x,info_y,cv=cv,scoring="accuracy",n_jobs=-1))
```

Out[67]:
```
0.732361041357044
```

Random Forest

In [40]:
```
rdf_params={"criterion":["entropy"],#gini
           "max_depth":[8],  #2,4,6
           "min_samples_split":[4],#2,6
           "max_features":["sqrt"]} #log2
```

In [41]:
```
rdf=RandomForestClassifier()
```

In [42]:
```
rdf_grid=GridSearchCV(rdf,rdf_params,cv=cv,verbose=2,n_jobs=-1)
```

In [43]:
```
rdf_grid.fit(info_x,info_y)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    3.9s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   27.0s
[Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:   35.9s finished
```

Out[43]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=2020, shuffle=False),
             estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 6, 8],
                         'max_features': ['sqrt', 'log2'],
                         'min_samples_split': [2, 4, 6]},
             verbose=2)
```

In [44]:
```
rdf_grid.best_params_
```

Out[44]:
```
{'criterion': 'entropy',
 'max_depth': 8,
 'max_features': 'sqrt',
 'min_samples_split': 4}
```

In [447]:
```
rdf=RandomForestClassifier(criterion="entropy",max_depth=8,max_features="sqrt",min_samples
```

In [47]:
```
np.mean(cross_val_score(rdf,info_x,info_y,cv=cv,scoring="accuracy",n_jobs=-1))
```

Out[47]: 0.7297290524265874

XGBClassifier

In [68]:
```
xgb_params = {"eta":[0.01,0.05,0.1],
              "max_depth":[2,4,6,8],
              "objective":["binary:logistic"],
             "lambda":[1.20,50],
             "alpha":[0,20],
             "subsample":[0.3,0.7,0.8,1]}
```

In [69]:
```
xgb=XGBClassifier()
```

In [70]:
```
xgb_grid=GridSearchCV(xgb,xgb_params,cv=cv,verbose=2,n_jobs=-1)
```

In [71]:
```
xgb_grid.fit(info_x,info_y)
```

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   11.6s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  2.4min
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 960 out of 960 | elapsed:  6.6min finished
```

Out[71]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=2020, shuffle=False),
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, m...
                                     n_estimators=100, n_jobs=None,
                                     num_parallel_tree=None, random_state=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=None, subsample=None,
                                     tree_method=None, validate_parameters=None,
                                     verbosity=None),
             n_jobs=-1,
             param_grid={'alpha': [0, 20], 'eta': [0.01, 0.05, 0.1],
                         'lambda': [1.2, 50], 'max_depth': [2, 4, 6, 8],
                         'objective': ['binary:logistic'],
                         'subsample': [0.3, 0.7, 0.8, 1]},
             verbose=2)
```

In [72]:
```
xgb_grid.best_params_
```

Out[72]:
```
{'alpha': 0,
 'eta': 0.05,
 'lambda': 50,
 'max_depth': 2,
 'objective': 'binary:logistic',
 'subsample': 1}
```

In [448]:
```
xgb=XGBClassifier(eta=0.05,max_depth=2,objective="binary:logistic",reg_lambda=50)
```

Ensemble, Stacking

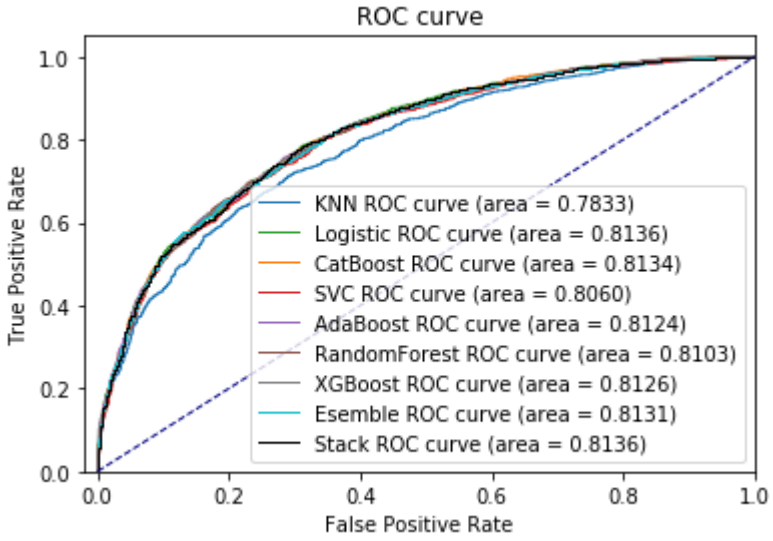
In [424]:
```
eclf = EnsembleVoteClassifier(clfs=[cat,logreg, knn, svc,ada,rdf,xgb], weights=[1,1,1,1,1,
labels = ['CatBoost','Logistic Regression', 'KNN', 'SVC','AdaBoost',"Random Forest",'XGBoo
for clf, label in zip([cat,logreg, knn, svc, ada, rdf, xgb,eclf], labels):

    scores = cross_val_score(clf, info_x, info_y,
                             cv=cv,
                             scoring='accuracy',
                             n_jobs=-1)

    print("[%s] Accuracy: %0.6f (+/- %0.6f) Best: %0.6f "
          % (label,scores.mean(), scores.std(), scores.max()))
```

```
[CatBoost] Accuracy: 0.729832 (+/- 0.010683) Best: 0.739372
[Logistic Regression] Accuracy: 0.730641 (+/- 0.009936) Best: 0.739879
[KNN] Accuracy: 0.706853 (+/- 0.013521) Best: 0.732794
[SVC] Accuracy: 0.721227 (+/- 0.012242) Best: 0.735324
[AdaBoost] Accuracy: 0.732767 (+/- 0.011828) Best: 0.744433
[Random Forest] Accuracy: 0.728313 (+/- 0.007885) Best: 0.738866
[XGBoost] Accuracy: 0.733374 (+/- 0.010998) Best: 0.746964
[Ensemble] Accuracy: 0.730237 (+/- 0.009251) Best: 0.739372
```

In [467]:
```
X_train, X_test, y_train, y_test = train_test_split(info_x,info_y,random_state=666,test_si
```

In [468]:
```
plt.figure()
lw = 1
#knn
knn.fit(X_train,y_train)
knn_pred = knn.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,knn_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:blue',
             lw=lw, label='KNN ROC curve (area = %0.4f)' % roc_auc)
#logreg
logreg.fit(X_train,y_train)
log_pred = logreg.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,log_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:green',
             lw=lw, label='Logistic ROC curve (area = %0.4f)' % roc_auc)

#cat
cat.fit(X_train,y_train)
cat_pred = cat.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,cat_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:orange',
             lw=lw, label='CatBoost ROC curve (area = %0.4f)' % roc_auc)

#svc
svc.fit(X_train,y_train)
svc_pred = svc.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,svc_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:red',
             lw=lw, label='SVC ROC curve (area = %0.4f)' % roc_auc)

#ada
ada.fit(X_train,y_train)
ada_pred = ada.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,ada_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:purple',
             lw=lw, label='AdaBoost ROC curve (area = %0.4f)' % roc_auc)

#rdf
rdf.fit(X_train,y_train)
rdf_pred = rdf.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,rdf_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:brown',
             lw=lw, label='RandomForest ROC curve (area = %0.4f)' % roc_auc)

#xgb
xgb.fit(X_train,y_train)
xgb_pred = xgb.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,xgb_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:gray',
             lw=lw, label='XGBoost ROC curve (area = %0.4f)' % roc_auc)
```

```
#eclf
eclf.fit(X_train,y_train)
eclf_pred = eclf.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,eclf_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='tab:cyan',
             lw=lw, label='Esemble ROC curve (area = %0.4f)' % roc_auc)

#stack
stack.fit(X_train,y_train)
stack_pred = stack.predict_proba(X_test)
fpr, tpr, threshold = roc_curve(y_test,stack_pred[:,1])
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='black',
             lw=lw, label='Stack ROC curve (area = %0.4f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([-0.02, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
plt.legend(loc="lower right")
plt.show()
```



In [78]:
```
estimators = [
    ('knn', KNeighborsClassifier(algorithm= 'ball_tree', n_neighbors=98, weights= 'distanc
    ('rdf', RandomForestClassifier(criterion="entropy",max_depth=8,max_features="sqrt",mir
    ('svc',SVC(kernel="poly",probability=True,C=20)),
    ('cat',CatBoostClassifier(learning_rate=0.005,depth=8,verbose=False)),
    ('ada',AdaBoostClassifier(learning_rate=0.05,algorithm="SAMME.R",n_estimators=150)),
    ('logreg',LogisticRegression(penalty="l2",solver="sag",warm_start=False,C=50)),
    ('xgb',XGBClassifier(eta=0.05,max_depth=2,objective="binary:logistic",reg_lambda=50))
]


stack = StackingClassifier(estimators=estimators,
                           final_estimator=LogisticRegression(penalty="l2",solver="sag",wa
```

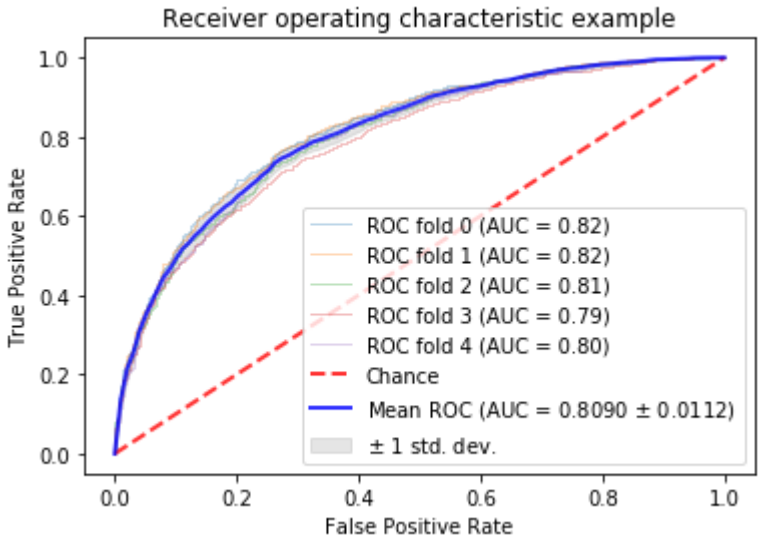
In [79]:
```
stack_score = cross_val_score(stack, info_x, info_y,cv=cv,scoring='accuracy',n_jobs=-1)
```

Out[79]: 0.7321588171987906

In [89]:
```
print(np.mean(stack_score),np.std(stack_score))
print(stack_score.max())
```

```
0.7321588171987906 0.005170871168699806
0.7388663967611336
```

In [440]:
```
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax = plt.subplots()
for i, (train, test) in enumerate(cv.split(info_x, info_y)):
    stack.fit(info_x.iloc[train], info_y.iloc[train])
    viz = plot_roc_curve(stack, info_x.iloc[test], info_y.iloc[test],
                         name='ROC fold {}'.format(i),
                         alpha=0.3, lw=1, ax=ax)
    interp_tpr = np.interp(mean_fpr, viz.fpr, viz.tpr)
    interp_tpr[0] = 0.0
    tprs.append(interp_tpr)
    aucs.append(viz.roc_auc)

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.8)

mean_tpr = np.mean(tprs, axis=0)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
std_auc = np.std(aucs)
ax.plot(mean_fpr, mean_tpr, color='b',
        label=r'Mean ROC (AUC = %0.4f $\pm$ %0.4f)' % (mean_auc, std_auc),
        lw=2, alpha=.8)

std_tpr = np.std(tprs, axis=0)
tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
ax.fill_between(mean_fpr, tprs_lower, tprs_upper, color='grey', alpha=.2,
                label=r'$\pm$ 1 std. dev.')

ax.set(xlim=[-0.05, 1.05], ylim=[-0.05, 1.05],
       title="Receiver operating characteristic example")
ax.legend(loc="lower right")
plt.show()
```



False Prediction Analysis

In [469]:
```
stack.fit(X_train,y_train)
stack_pred = stack.predict_proba(X_test)
```

In [470]:
```
X_test["blueWins"]=y_test
X_test["Prediction"]=stack_pred[:,1].round().astype(int)
```

In [471]:
```
X_test["Error"] = 3
for i in X_test.index:
    if X_test["blueWins"][i] != X_test["Prediction"][i]:
        X_test["Error"][i] = 1
    else:
        X_test["Error"][i] = 0
```

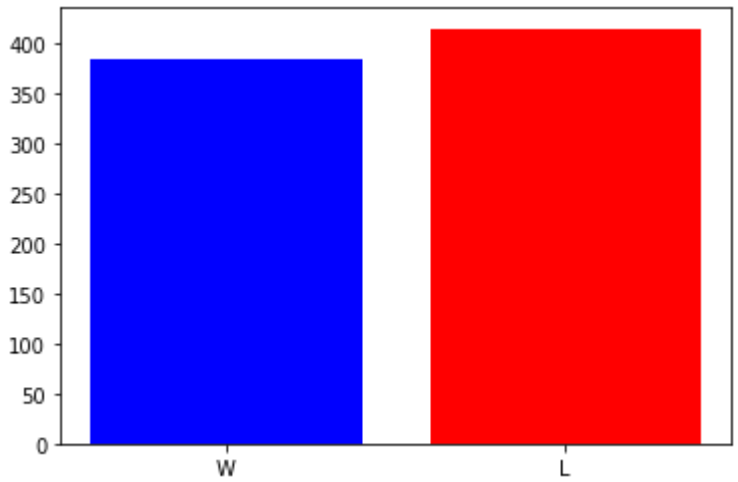
In [473]:
```
X_test.Error.value_counts()
```

Out[473]:
```
0    2165
1     799
Name: Error, dtype: int64
```

In [474]:
```
Error = X_test.where(X_test["Error"]==1).dropna()
Correct = X_test.where(X_test["Error"]==0).dropna()
Error_normal = data.loc[Error.index]
Correct_normal = data.loc[Correct.index]
data_normal = data.loc[X_test.index]
```

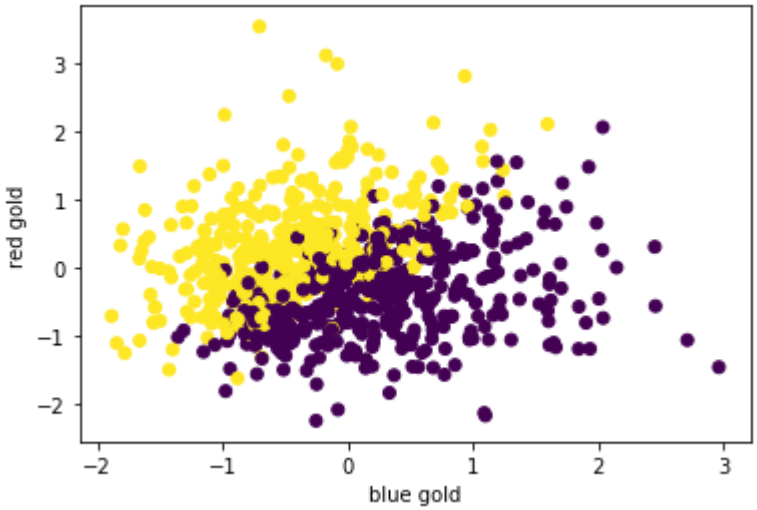
In [475]:
```
plt.bar(Error.blueWins.unique(),Error.blueWins.value_counts(),
        color=("r","b"),tick_label=("L","W"))
print(Error.blueWins.value_counts(normalize=True))
```

```
0.0    0.519399
1.0    0.480601
Name: blueWins, dtype: float64
```



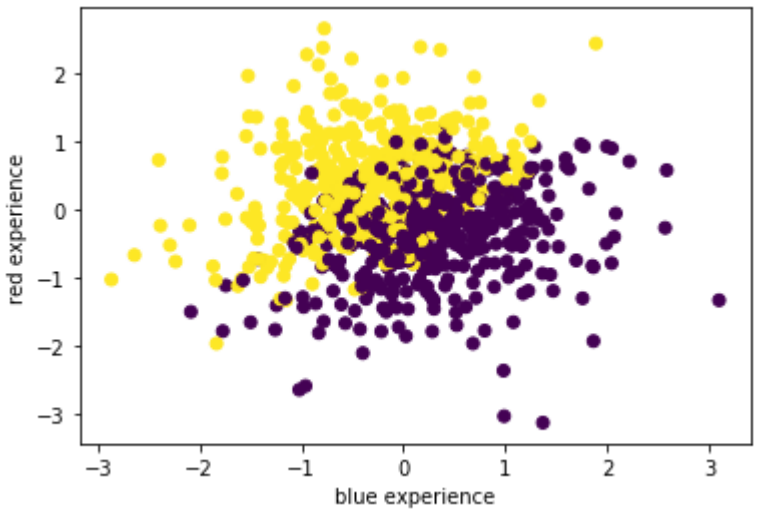
In [476]:
```
plt.scatter(Error.blueTotalGold,Error.redTotalGold,c=Error.blueWins)
plt.xlabel("blue gold")
plt.ylabel("red gold")
```

Out[476]: Text(0, 0.5, 'red gold')



In [477]:
```
plt.scatter(Error.blueTotalExperience, Error.redTotalExperience, c=Error.blueWins)
plt.xlabel("blue experience")
plt.ylabel("red experience")
```

Out[477]: Text(0, 0.5, 'red experience')



In [478]:
```
Error_normal
```

Out[478]:

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	bluel
4433	4516589637	1	14	1	0	9	
1249	4517512393	1	14	4	1	9	
3860	4522690291	1	12	1	1	7	
7547	4486105297	1	18	2	1	7	
6729	4477434796	0	17	3	0	8	
...	...	...	...	...	...	...	
7022	4519668670	0	136	4	1	9	
6334	4487018877	1	16	4	1	5	
905	4516044973	0	13	0	0	5	
9241	4505233035	1	39	0	1	5	
1457	4456960285	1	19	5	0	10	

799 rows × 40 columns

In [479]:
```
print("Early JGDiff Avg: %0.4f"
      %(data.blueTotalJungleMinionsKilled-data.redTotalJungleMinionsKilled).mean())
print("Early JGDiff Avg for the holdout: %0.4f"
      %(data_normal.blueTotalJungleMinionsKilled-data_normal.redTotalJungleMinionsKilled).
print("Early JGDiff Avg for Wrong Prediction: %0.4f"
      %(Error_normal.blueTotalJungleMinionsKilled-Error_normal.redTotalJungleMinionsKilled
print("Early JGDiff Avg for Correct Prediction: %0.4f"
      %(Correct_normal.blueTotalJungleMinionsKilled-Correct_normal.redTotalJungleMinionsKi
```

```
Early JGDiff Avg: -0.8034
Early JGDiff Avg for the holdout: -1.0067
Early JGDiff Avg for Wrong Prediction: -0.9837
Early JGDiff Avg for Correct Prediction: -1.0152
```

In [ ]:
```

```