In [1]:
```
import numpy as np
import pandas as pd
pd.options.display.max_columns=100
from sklearn.model_selection import train_test_split, cross_val_score, cr
oss_validate
from sklearn.preprocessing import StandardScaler as SSc
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.neighbors import KNeighborsClassifier as KNC
import matplotlib.pyplot as plt
import graphviz as gviz
%matplotlib inline

#set width of window to preference
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
```

In [2]:
```
#year = "2019"                                                    #choos
e year to get data from
#split = "summer"                                                 #choos
e split to get data from(spring, summer, worlds)
#infile = r"C:\Users\Triplea657\000 MSCS-335 2020\Datasets\League_"#path
#inf = "-Wrangled.csv"                                            #file
to read
#filein = infile+year+"\\"+year+'-'+split+'-'+inf
#data = pd.read_csv(filein,low_memory=False)
#data.head(10)

#changed for submission version
data = pd.read_csv("Datasets/League_2019/2019-summer-Wrangled.csv", index
_col=0, low_memory=False)
data.head()
```

Out[2]:

	league_CBLoL	league_LCK	league_LCS	league_LEC	league_LMS	gamelength	result	k
0	0.0	0.0	1.0	0.0	0.0	35.500000	1.0	21.0
1	0.0	0.0	1.0	0.0	0.0	35.500000	0.0	14.0
2	0.0	0.0	1.0	0.0	0.0	29.700000	1.0	11.0
3	0.0	0.0	1.0	0.0	0.0	29.700000	0.0	4.0
4	0.0	0.0	1.0	0.0	0.0	31.983333	1.0	12.0

1 - Trying to predict the region in which the game was played

Split data into X,Y where Y is which region the game was played in, then normalize the inputs.

In [3]:
```
X = data.iloc[:,5:]
Y = data.iloc[:,:5]
#transform input data (normalize)
ssc = SSc()
Xft = ssc.fit_transform(X)
X = pd.DataFrame(Xft)
print("Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes: ")
Xtr,Xtst,Ytr,Ytst = train_test_split(X,Y,test_size=0.2,random_state=2020)
print(Xtr.shape,Xtst.shape,Ytr.shape,Ytst.shape)
```

```
Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes:
(1155, 57) (289, 57) (1155, 5) (289, 5)
```

Create and train models on training data then check their accuracy on test data to check accuracy of region prediction

In [4]:
```
print("Classifier scores:\n"+"-"*18+"\n")

tree = DTC(max_depth=5)
tree.fit(Xtr,Ytr)
scr = cross_val_score(tree,Xtst,Ytst, cv=5)
print("Tree \nscore avg:"+str(sum(scr)/5)+"\nscore = "+str(scr)+"\n\
n"+"-"*64)

for i in range(1,20,2):
    forest = RFC(n_estimators=i,max_depth=5)
    forest.fit(Xtr,Ytr)
    scr = cross_val_score(forest,Xtst,Ytst, cv=5)
    print("\nRandom Forest trees = "+str(i)+" depth = 5 \nscore avg: "+st
r(sum(scr)/5)+" \nscores: "+str(scr))

print("\n"+"-"*64)

for i in range(1,16,3):
    knn = KNC(n_neighbors=i)
    knn.fit(Xtr,Ytr)
    scr = cross_val_score(knn,Xtst,Ytst, cv=5)
    print("\nK-Nearest Neighbors "+str(i)+"-neighbors\nscore avg:"+str(su
m(scr)/5)+"\nscore = "+
          str(scr))
```

```
Classifier scores:
------------------

Tree
score avg:0.28021778584392015
score = [0.31034483 0.32758621 0.27586207 0.22413793 0.26315789]

---------------------------------------------------------------

Random Forest trees = 1 depth = 5
score avg: 0.19709618874773138
scores: [0.13793103 0.25862069 0.15517241 0.27586207 0.15789474]

Random Forest trees = 3 depth = 5
score avg: 0.2004839685420448
scores: [0.15517241 0.20689655 0.22413793 0.27586207 0.14035088]

Random Forest trees = 5 depth = 5
score avg: 0.14875983061101028
scores: [0.18965517 0.15517241 0.12068966 0.13793103 0.14035088]

Random Forest trees = 7 depth = 5
score avg: 0.16957047791893526
scores: [0.18965517 0.18965517 0.15517241 0.13793103 0.1754386 ]

Random Forest trees = 9 depth = 5
score avg: 0.14875983061101028
scores: [0.20689655 0.12068966 0.13793103 0.13793103 0.14035088]

Random Forest trees = 11 depth = 5
score avg: 0.12462189957652751
scores: [0.17241379 0.15517241 0.10344828 0.05172414 0.14035088]

Random Forest trees = 13 depth = 5
score avg: 0.1730792498487598
scores: [0.18965517 0.18965517 0.13793103 0.15517241 0.19298246]

Random Forest trees = 15 depth = 5
score avg: 0.17301875378100423
scores: [0.17241379 0.22413793 0.15517241 0.13793103 0.1754386 ]

Random Forest trees = 17 depth = 5
score avg: 0.17301875378100423
scores: [0.18965517 0.24137931 0.15517241 0.10344828 0.1754386 ]

Random Forest trees = 19 depth = 5
score avg: 0.1453720508166969
scores: [0.17241379 0.13793103 0.12068966 0.13793103 0.15789474]

---------------------------------------------------------------

K-Nearest Neighbors 1-neighbors
score avg:0.3254083484573502
score = [0.29310345 0.43103448 0.25862069 0.27586207 0.36842105]

K-Nearest Neighbors 4-neighbors
score avg:0.15238959467634605
score = [0.17241379 0.17241379 0.10344828 0.12068966 0.19298246]

K-Nearest Neighbors 7-neighbors
score avg:0.18348457350272232
score = [0.24137931 0.22413793 0.13793103 0.10344828 0.21052632]

K-Nearest Neighbors 10-neighbors
```

This shows that the region in which the games are played is extremely difficult to predict (at least with these methods so far). After playing with the numbers a bit it becomes clear that a single tree of depth 5 far outperforms the other methods chosen. This likely indicates that it is difficult to predict the region as the trends aren't strong enough to definitively classify a game as being played in a particular region. Further analysis may reveal that this could be due to some regions being similar and thus difficult to distinguish or it could simply be that the data is too similar across all regions meaning that all attempts to predict region of play will be unsuccessful.

---

2 - Trying to predict the result of the game played

Split data into X,Y where Y is whether the current team won the game, then normalize the inputs.

In [5]:
```
X = data.iloc[:,data.columns != 'result']
Y = data.iloc[:,data.columns == 'result']
#transform input data (normalize)
ssc = SSc()
Xft = ssc.fit_transform(X)
X = pd.DataFrame(Xft)
print("Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes: ")
Xtr,Xtst,Ytr,Ytst = train_test_split(X,Y.values.ravel(),test_size=0.2,ran
dom_state=2020)
print(Xtr.shape,Xtst.shape,Ytr.shape,Ytst.shape)
```

```
Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes:
(1155, 61) (289, 61) (1155,) (289,)
```

Create and train models on training data then check their accuracy on test data to check accuracy of game score prediction

In [6]:
```
print("Classifier scores:\n")

tree = DTC()
tree.fit(Xtr,Ytr)
scr = cross_val_score(tree,Xtst,Ytst, cv=5)
print("Tree \nscore avg: "+str(sum(scr)/5)+"\nscore = "+str(scr)+"\n\
n"+"-"*64)

for i in range(1,202,40):
    forest = RFC(n_estimators=i,max_depth=5)
    forest.fit(Xtr,Ytr)
    scr = cross_val_score(forest,Xtst,Ytst, cv=5)
    print("\nRandom Forest trees = "+str(i)+" depth = 5 \nscore avg: "+st
r(sum(scr)/5)+" \nscores: "+str(scr))

print("\n"+"-"*64)

for i in range(1,16,5):
    knn = KNC(n_neighbors=i)
    knn.fit(Xtr,Ytr)
    scr = cross_val_score(knn,Xtst,Ytst, cv=5)
    print("\nK-Nearest Neighbors "+str(i)+"-neighbors\nscore avg:"+str(su
m(scr)/5)+"\nscore = "+str(scr))
```

```
Classifier scores:

Tree
score avg: 0.941016333938294
score = [0.94827586 0.94827586 0.93103448 0.98275862 0.89473684]

---------------------------------------------------------------

Random Forest trees = 1 depth = 5
score avg: 0.8855414398064125
scores: [0.89655172 0.93103448 0.9137931  0.87931034 0.80701754]

Random Forest trees = 41 depth = 5
score avg: 0.9549304295220811
scores: [1.         0.98275862 0.89655172 0.96551724 0.92982456]

Random Forest trees = 81 depth = 5
score avg: 0.9548699334543255
scores: [1.         0.96551724 0.93103448 0.96551724 0.9122807 ]

Random Forest trees = 121 depth = 5
score avg: 0.951361161524501
scores: [1.         0.96551724 0.9137931  0.98275862 0.89473684]

Random Forest trees = 161 depth = 5
score avg: 0.9548699334543255
scores: [1.         0.98275862 0.89655172 0.98275862 0.9122807 ]

Random Forest trees = 201 depth = 5
score avg: 0.9583182093163944
scores: [1.         0.98275862 0.9137931  0.98275862 0.9122807 ]

---------------------------------------------------------------

K-Nearest Neighbors 1-neighbors
score avg:0.8924984875983062
score = [0.9137931  0.9137931  0.89655172 0.9137931  0.8245614 ]

K-Nearest Neighbors 6-neighbors
score avg:0.9065335753176044
score = [0.94827586 0.93103448 0.9137931  0.84482759 0.89473684]

K-Nearest Neighbors 11-neighbors
score avg:0.9341802782819115
score = [0.94827586 0.94827586 0.93103448 0.93103448 0.9122807 ]
```

In [7]:
```
#
-----------------------------------------------------------------------
-----------------------------------
#---begin code
#
-----------------------------------------------------------------------
-----------------------------------
breakline = "-"*64

scoring = {'FVE': 'explained_variance',
           'MSE': 'neg_mean_squared_error',
           'R2': 'r2'}
def cscore(model,X,Y):
    cr_v = cross_validate(model, X, Y, scoring=scoring,cv=5, return_train
_score=False)
    return cr_v
def test(X,Y):
    for i in range(2):
        dpth = 3+i
        tree = DTC(max_depth=dpth)
        tree.fit(Xtr,Ytr)
        scr = cscore(tree,Xtst,Ytst)
        for j,k in enumerate(scr.keys()):
            if j > 1:
                if(k=='test_MSE'):
                    print("-----{} (0.0 is best)\nscores:   {}\navg scor
e: {}".format(k,scr[k],scr[k].mean()))
                else:
                    print("-----{} (1.0 is best)\nscores:   {}\navg scor
e: {}".format(k,scr[k],scr[k].mean()))



    print("\n\n"+"-"*36)

    for l in range(3,12,2):
        knn = KNC(n_neighbors=l)
        knn.fit(Xtr,Ytr)
        scr = cscore(knn,Xtst,Ytst)
        print("\n"+"-"*10+"K-Nearest Neighbors, {}-neighbors scores:".for
mat(l))
        for j,k in enumerate(scr.keys()):
            if j > 1:
                if(k=='test_MSE'):
                    print("-----{} (0.0 is best)\nscores:   {}\navg scor
e: {}".format(k,scr[k],scr[k].mean()))
                else:
                    print("-----{} (1.0 is best)\nscores:   {}\navg scor
e: {}".format(k,scr[k],scr[k].mean()))


    print("\n\n"+"-"*36)

    dpth=4
    for l in range(1,16,5): #change number of trees
        forest = RFC(n_estimators=l,max_depth=dpth)
        forest.fit(Xtr,Ytr)
        scr = cscore(forest,Xtst,Ytst)
        print("\n"+"-"*10+"Random Forest, {} trees of depth {} scores:".f
ormat(l,dpth))
        for j,k in enumerate(scr.keys()):
            if j > 1:
                if(k=='test_MSE'):
```

```
FVE best score: 1.0
MSE best score: 0.0, negative indicates that 0.0 is the best score as o
pposed to 1.0
FVE best score: 1.0
-----test_FVE (1.0 is best)
scores:   [0.74285714 0.65595238 0.72759857 0.79330944 0.64938272]
avg score: 0.7138200489275758
-----test_MSE (0.0 is best)
scores:   [-0.06896552 -0.0862069  -0.06896552 -0.05172414 -0.0877193 ]
avg score: -0.07271627344222625
-----test_R2 (1.0 is best)
scores:   [0.72380952 0.6547619  0.72281959 0.7921147  0.64814815]
avg score: 0.7083307731694829
-----test_FVE (1.0 is best)
scores:   [0.86666667 0.72380952 0.72759857 0.79330944 0.71851852]
avg score: 0.7659805427547364
-----test_MSE (0.0 is best)
scores:   [-0.03448276 -0.06896552 -0.06896552 -0.05172414 -0.07017544]
avg score: -0.0588626739261948
-----test_R2 (1.0 is best)
scores:   [0.86190476 0.72380952 0.72281959 0.7921147  0.71851852]
avg score: 0.7638334186721284


-------------------------------------

----------K-Nearest Neighbors, 3-neighbors scores:
-----test_FVE (1.0 is best)
scores:   [0.86666667 0.66547619 0.72281959 0.58900836 0.71851852]
avg score: 0.7124978665301246
-----test_MSE (0.0 is best)
scores:   [-0.03448276 -0.0862069  -0.06896552 -0.10344828 -0.07017544]
avg score: -0.07265577737447065
-----test_R2 (1.0 is best)
scores:   [0.86190476 0.6547619  0.72281959 0.58422939 0.71851852]
avg score: 0.708446833930705

----------K-Nearest Neighbors, 5-neighbors scores:
-----test_FVE (1.0 is best)
scores:   [0.80357143 0.79404762 0.65471924 0.52568698 0.57777778]
avg score: 0.6711606076122206
-----test_MSE (0.0 is best)
scores:   [-0.05172414 -0.05172414 -0.0862069  -0.12068966 -0.10526316]
avg score: -0.08312159709618874
-----test_R2 (1.0 is best)
scores:   [0.79285714 0.79285714 0.65352449 0.51493429 0.57777778]
avg score: 0.666390168970814

----------K-Nearest Neighbors, 7-neighbors scores:
-----test_FVE (1.0 is best)
scores:   [0.80357143 0.86190476 0.58422939 0.65471924 0.57777778]
avg score: 0.6964405188598737
-----test_MSE (0.0 is best)
scores:   [-0.05172414 -0.03448276 -0.10344828 -0.0862069  -0.10526316]
avg score: -0.07622504537205081
-----test_R2 (1.0 is best)
scores:   [0.79285714 0.86190476 0.58422939 0.65352449 0.57777778]
avg score: 0.694058713090971

----------K-Nearest Neighbors, 9-neighbors scores:
-----test_FVE (1.0 is best)
scores:   [0.65595238 0.79404762 0.72281959 0.72281959 0.58271605]
avg score: 0.6956710473914777
-----test_MSE (0.0 is best)
```

It is clear that the tree models perform incredibly well in predicting the victory or defeat of a team and that when making a forest only occasionally does there exist a tree that fails to completely accurately classify the match as a victory or a loss. The K-nearest neighbors algorithm also classifies this very well.

3 - Trying to predict whether current team was the first to take 3 towers

Split data into X,Y where Y is whether the current team was the first to get to 3 tower kills, then normalize the inputs.

In [8]:
```
X = data.iloc[:,data.columns != 'firsttothreetowers']
Y = data.iloc[:,data.columns == 'firsttothreetowers']
#transform input data (normalize)
ssc = SSc()
Xft = ssc.fit_transform(X)
X = pd.DataFrame(Xft)
print("Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes: ")
Xtr,Xtst,Ytr,Ytst = train_test_split(X,Y.values.ravel(),test_size=0.2,ran
dom_state=2020)
print(Xtr.shape,Xtst.shape,Ytr.shape,Ytst.shape)
```

```
Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes:
(1155, 61) (289, 61) (1155,) (289,)
```

In [9]:
```
print("Classifier scores:\n")

tree = DTC()
tree.fit(Xtr,Ytr)
scr = cross_val_score(tree,Xtst,Ytst, cv=5)
print("Tree \nscore avg: "+str(sum(scr)/5)+"\nscore = "+str(scr)+"\n\
n"+"-"*64)

for i in range(1,20,4):
    dpth = 4
    forest = RFC(n_estimators=i,max_depth=dpth)
    forest.fit(Xtr,Ytr)
    scr = cross_val_score(forest,Xtst,Ytst, cv=5)
    print("\nRandom Forest trees = "+str(i)+" depth = "+str(dpth)+" \nsco
re avg: "+str(sum(scr)/5)+" \nscores: "+str(scr))

print("\n"+"-"*64)

for i in range(1,21,4):
    knn = KNC(n_neighbors=i)
    knn.fit(Xtr,Ytr)
    scr = cross_val_score(knn,Xtst,Ytst, cv=5)
    print("\nK-Nearest Neighbors "+str(i)+"-neighbors\nscore avg:"+str(su
m(scr)/5)+"\nscore = "+str(scr))
```

```
Classifier scores:

Tree
score avg: 0.8649122807017544
score = [0.86206897 0.82758621 0.87931034 0.93103448 0.8245614 ]

----------------------------------------------------------------

Random Forest trees = 1 depth = 4
score avg: 0.8099213551119178
scores: [0.74137931 0.9137931  0.75862069 0.75862069 0.87719298]

Random Forest trees = 5 depth = 4
score avg: 0.8580157289776166
scores: [0.82758621 0.9137931  0.86206897 0.86206897 0.8245614 ]

Random Forest trees = 9 depth = 4
score avg: 0.8822141560798548
scores: [0.89655172 0.94827586 0.89655172 0.82758621 0.84210526]

Random Forest trees = 13 depth = 4
score avg: 0.8718088324258924
scores: [0.86206897 0.94827586 0.89655172 0.82758621 0.8245614 ]

Random Forest trees = 17 depth = 4
score avg: 0.8546884452510586
scores: [0.79310345 0.9137931  0.87931034 0.82758621 0.85964912]

----------------------------------------------------------------

K-Nearest Neighbors 1-neighbors
score avg:0.8304900181488204
score = [0.81034483 0.81034483 0.84482759 0.84482759 0.84210526]

K-Nearest Neighbors 5-neighbors
score avg:0.8439806412583183
score = [0.82758621 0.89655172 0.89655172 0.84482759 0.75438596]

K-Nearest Neighbors 9-neighbors
score avg:0.8440411373260737
score = [0.86206897 0.86206897 0.9137931  0.81034483 0.77192982]

K-Nearest Neighbors 13-neighbors
score avg:0.8406533575317605
score = [0.84482759 0.89655172 0.87931034 0.79310345 0.78947368]

K-Nearest Neighbors 17-neighbors
score avg:0.8406533575317605
score = [0.82758621 0.87931034 0.87931034 0.82758621 0.78947368]
```

Whether the present team was the first to capture 3 towers is also a fairly high accuracy test. The random forest with a fairly large number of trees performed slightly better than a single tree or any number of neighbors. The forest consistently reaches near-peak accuracy with 10 trees and having no significant gains with more.

4 - Trying to predict whether the current team was the first to take three towers without as input

Split data into X,Y where Y is whether the current team was the first to get to 3 tower kills, then normalize the inputs.

In [10]:
```
X = data.drop(['firstmidouter','firsttothreetowers','result','teambaronkills'],axis=1)
Y = data.iloc[:,data.columns == 'firsttothreetowers']
#transform input data (normalize)
ssc = SSc()
Xft = ssc.fit_transform(X)
X = pd.DataFrame(Xft)
print("Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes: ")
Xtr,Xtst,Ytr,Ytst = train_test_split(X,Y.values.ravel(),test_size=0.2,random_state=2020)
print(Xtr.shape,Xtst.shape,Ytr.shape,Ytst.shape)
```

```
Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes:
(1155, 58) (289, 58) (1155,) (289,)
```

In [11]:
```
print("Classifier scores:\n")

tree = DTC()
tree.fit(Xtr,Ytr)
scr = cross_val_score(tree,Xtst,Ytst, cv=5)
print("Tree \nscore avg: "+str(sum(scr)/5)+"\nscore = "+str(scr)+"\n\
n"+"-"*64)

for i in range(1,20,4):
    dpth = 4
    forest = RFC(n_estimators=i,max_depth=dpth)
    forest.fit(Xtr,Ytr)
    scr = cross_val_score(forest,Xtst,Ytst, cv=5)
    print("\nRandom Forest trees = "+str(i)+" depth = "+str(dpth)+" \nsco
re avg: "+str(sum(scr)/5)+" \nscores: "+str(scr))

print("\n"+"-"*64)

for i in range(1,21,4):
    knn = KNC(n_neighbors=i)
    knn.fit(Xtr,Ytr)
    scr = cross_val_score(knn,Xtst,Ytst, cv=5)
    print("\nK-Nearest Neighbors "+str(i)+"-neighbors\nscore avg:"+str(su
m(scr)/5)+"\nscore = "+str(scr))
```

```
Classifier scores:

Tree
score avg: 0.7679975801572898
score = [0.70689655 0.84482759 0.81034483 0.75862069 0.71929825]

---------------------------------------------------------------

Random Forest trees = 1 depth = 4
score avg: 0.7648517846339987
scores: [0.75862069 0.75862069 0.79310345 0.70689655 0.80701754]

Random Forest trees = 5 depth = 4
score avg: 0.8061705989110708
scores: [0.79310345 0.87931034 0.82758621 0.74137931 0.78947368]

Random Forest trees = 9 depth = 4
score avg: 0.8199032062915912
scores: [0.81034483 0.87931034 0.86206897 0.77586207 0.77192982]

Random Forest trees = 13 depth = 4
score avg: 0.8408348457350272
scores: [0.82758621 0.9137931  0.84482759 0.77586207 0.84210526]

Random Forest trees = 17 depth = 4
score avg: 0.8338777979431338
scores: [0.84482759 0.9137931  0.84482759 0.74137931 0.8245614 ]

---------------------------------------------------------------

K-Nearest Neighbors 1-neighbors
score avg:0.8030248033877798
score = [0.77586207 0.79310345 0.77586207 0.79310345 0.87719298]

K-Nearest Neighbors 5-neighbors
score avg:0.837265577737447
score = [0.79310345 0.87931034 0.86206897 0.84482759 0.80701754]

K-Nearest Neighbors 9-neighbors
score avg:0.8267392619479734
score = [0.81034483 0.87931034 0.87931034 0.81034483 0.75438596]

K-Nearest Neighbors 13-neighbors
score avg:0.8337568058076226
score = [0.81034483 0.9137931  0.86206897 0.79310345 0.78947368]

K-Nearest Neighbors 17-neighbors
score avg:0.8269207501512403
score = [0.79310345 0.87931034 0.84482759 0.81034483 0.80701754]
```

This shows that the results from part 3 were not due to using the result of the game and assuming that whoever won was the first to take 3 towers, but rather actually predicting whether the towers were taken based on the game statisics.