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
import sklearn.metrics
from sklearn.preprocessing import StandardScaler as SSc
from sklearn.tree import DecisionTreeRegressor as DTR
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.neighbors import KNeighborsRegressor as KNR
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>"))
display(HTML("<style>.output.output_scroll{ height:100% !important; }</st
yle>")) #breaks scroll output vertical so you see the whole output, disab
le this if you prefer.
```

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

In [3]:
```
var = []
for i in data:
    var.append(i)
print(var)
for i, v in enumerate(var):
    print(i, v)
```

```
['league_CBLoL', 'league_LCK', 'league_LCS', 'league_LEC', 'league_LMS
', 'gamelength', 'result', 'k', 'd', 'a', 'fb', 'kpm', 'okpm', 'ckpm',
'fd', 'fdtime', 'teamdragkills', 'oppdragkills', 'elementals', 'oppelem
entals', 'firedrakes', 'waterdrakes', 'earthdrakes', 'airdrakes', 'elde
rs', 'oppelders', 'herald', 'heraldtime', 'ft', 'fttime', 'firstmidoute
r', 'firsttothreetowers', 'teambaronkills', 'oppbaronkills', 'dmgtocham
ps', 'dmgtochampsperminute', 'wards', 'wpm', 'wardkills', 'wcpm', 'tota
lgold', 'earnedgpm', 'goldspent', 'gspd', 'monsterkillsownjungle', 'mon
sterkillsenemyjungle', 'cspm', 'goldat10', 'oppgoldat10', 'gdat10', 'go
ldat15', 'oppgoldat15', 'gdat15', 'xpat10', 'oppxpat10', 'xpdat10', 'cs
at10', 'oppcsat10', 'csdat10', 'csat15', 'oppcsat15', 'csdat15']
0 league_CBLoL
1 league_LCK
2 league_LCS
3 league_LEC
4 league_LMS
5 gamelength
6 result
7 k
8 d
9 a
10 fb
11 kpm
12 okpm
13 ckpm
14 fd
15 fdtime
16 teamdragkills
17 oppdragkills
18 elementals
19 oppelementals
20 firedrakes
21 waterdrakes
22 earthdrakes
23 airdrakes
24 elders
25 oppelders
26 herald
27 heraldtime
28 ft
29 fttime
30 firstmidouter
31 firsttothreetowers
32 teambaronkills
33 oppbaronkills
34 dmgtochamps
35 dmgtochampsperminute
36 wards
37 wpm
38 wardkills
39 wcpm
40 totalgold
41 earnedgpm
42 goldspent
43 gspd
44 monsterkillsownjungle
45 monsterkillsenemyjungle
46 cspm
47 goldat10
48 oppgoldat10
49 gdat10
50 goldat15
51 oppgoldat15
52 gdat15
```

1 - Trying to predict the length of the game played

Split data into X,Y where Y is the length of the game, then normalize the inputs.

In [4]:

```
X = data.iloc[:,data.columns != 'gamelength']
Y = data.iloc[:,data.columns == 'gamelength']
#transform input data (normalize scaling)
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, 61) (289, 61) (1155,) (289,)
```

In [5]:
```
print("Classifier scores:\n"+"-"*18+"\n")

tree = DTR(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 = RFR(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 = KNR(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:0.8688339362334357
score = [0.90064553 0.84418264 0.89714237 0.89148452 0.81071462]

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

Random Forest trees = 1 depth = 5
score avg: 0.8164661368933335
scores: [0.88549141 0.82110831 0.82165248 0.83091341 0.72316508]

Random Forest trees = 3 depth = 5
score avg: 0.9066697193082612
scores: [0.91099621 0.91309417 0.90988224 0.91149418 0.88788179]

Random Forest trees = 5 depth = 5
score avg: 0.894486939135383
scores: [0.94675132 0.89394793 0.8601876  0.88808896 0.88345888]

Random Forest trees = 7 depth = 5
score avg: 0.9155150617489675
scores: [0.90452363 0.92936666 0.90838157 0.91841461 0.91688883]

Random Forest trees = 9 depth = 5
score avg: 0.9167504015280432
scores: [0.95102155 0.93116634 0.88536546 0.92816278 0.88803588]

Random Forest trees = 11 depth = 5
score avg: 0.9187686643099913
scores: [0.93674206 0.92731609 0.90215989 0.9298727  0.89775258]

Random Forest trees = 13 depth = 5
score avg: 0.9263979910877467
scores: [0.94186259 0.93119973 0.90635365 0.92557475 0.92699923]

Random Forest trees = 15 depth = 5
score avg: 0.9273671560286475
scores: [0.93883081 0.93474997 0.91045486 0.92222491 0.93057523]

Random Forest trees = 17 depth = 5
score avg: 0.9272029254562095
scores: [0.94320471 0.93716017 0.90971531 0.92574404 0.9201904 ]

Random Forest trees = 19 depth = 5
score avg: 0.9285613689890869
scores: [0.9397332  0.93297646 0.91604547 0.92532215 0.92872957]

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

K-Nearest Neighbors 1-neighbors
score avg:0.5503541585117673
score = [0.43000924 0.71837904 0.50965417 0.59119001 0.50253834]

K-Nearest Neighbors 4-neighbors
score avg:0.7242329981047385
score = [0.68671451 0.79204271 0.76680979 0.71530448 0.6602935 ]

K-Nearest Neighbors 7-neighbors
score avg:0.7094477063937197
score = [0.67021646 0.74731298 0.74034743 0.68501081 0.70435085]

K-Nearest Neighbors 10-neighbors
```

K-Nearest Neighbors performs quite poorly, but trees perform fairly well and random forests perform very well even with fairly small numbers of trees.

---

2 - Trying to predict the number of kills, deaths and assists for the current team in the current game

Split data into X,Y where Y is the kills, deaths and assists, then normalize the inputs.

In [6]:
```
idx = [7,8,9]
idxtitles = ['kills', 'deaths', 'assists']
X = []
Y = []
for i in idx:
    X.append(data.drop(data.columns[idx],axis=1))
    Y.append(data.iloc[:,i])
    #transform input data (normalize scaling)
    ssc = SSc()
    Xft = ssc.fit_transform(X[i-7])
    X[i-7] = pd.DataFrame(Xft)
    print("Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes: ")
    Xtr,Xtst,Ytr,Ytst = train_test_split(X[i-7],Y[i-7],test_size=0.2,rand
om_state=2020)
    print(Xtr.shape,Xtst.shape,Ytr.shape,Ytst.shape)
```

```
Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes:
(1155, 59) (289, 59) (1155,) (289,)
Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes:
(1155, 59) (289, 59) (1155,) (289,)
Xtr(Xtrain),Xtst(Xtest),Ytr(Ytrain),Ytst(Ytest) shapes:
(1155, 59) (289, 59) (1155,) (289,)
```

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 tst(X,Y):
    for i in range(2):
        dpth = 3+i
        tree = DTR(max_depth=dpth)
        tree.fit(Xtr,Ytr)
        scr = cscore(tree,Xtst,Ytst)
        print("\nThe optimal depth is 3-4 for a single tree.\n"+"-"*10+"Tree of depth {} scores:".format(dpth))
        for j,k in enumerate(scr.keys()):
            if j > 1:
                if(k=='test_MSE'):
                    print("-----{} (0.0 is best)\nscores:   {}\navg score: {}".format(k,scr[k],scr[k].mean()))
                else:
                    print("-----{} (1.0 is best)\nscores:   {}\navg score: {}".format(k,scr[k],scr[k].mean()))




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

    print("\nK-Nearest Neighbors reaches near maximum accuracy at 5 neighbors and rising marginally until 9 neighbors")
    for l in range(3,12,2):
        knn = KNR(n_neighbors=l)
        knn.fit(Xtr,Ytr)
        scr = cscore(knn,Xtst,Ytst)
        print("\n"+"-"*10+"K-Nearest Neighbors, {}-neighbors scores:".format(l))
        for j,k in enumerate(scr.keys()):
            if j > 1:
                if(k=='test_MSE'):
                    print("-----{} (0.0 is best)\nscores:   {}\navg score: {}".format(k,scr[k],scr[k].mean()))
                else:
                    print("-----{} (1.0 is best)\nscores:   {}\navg score: {}".format(k,scr[k],scr[k].mean()))


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

    dpth=4
    print("\nRandom forests seem to reach a maximum accuracy of about 0.84 for FVE and R^2 with any parameters.\nIt reaches optimal accuracy most efficiently at ~10 trees of depth 3 and ~5 trees of depth 4.")
    for l in range(1,16,5): #change number of trees
        forest = RFR(n_estimators=l,max_depth=dpth)
```

```
I used a Decision Tree, K-Nearest Neighbors, and Random Forests to pred
ict ['kills', 'deaths', 'assists']

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

----------------------------------------------------------------
kills regressor scores:
----------------------------------------------------------------


The optimal depth is 3-4 for a single tree.
----------Tree of depth 3 scores:
-----test_FVE (1.0 is best)
scores:   [0.74296769 0.82130487 0.83242113 0.70568816 0.73976042]
avg score: 0.7684284535774695
-----test_MSE (0.0 is best)
scores:   [-57.98651027 -45.50304525 -39.9737957  -43.8362467  -64.0759
1737]
avg score: -50.27510305564451
-----test_R2 (1.0 is best)
scores:   [0.71647788 0.81876985 0.83023975 0.70354059 0.72240388]
avg score: 0.7582863895953074

The optimal depth is 3-4 for a single tree.
----------Tree of depth 4 scores:
-----test_FVE (1.0 is best)
scores:   [0.77095832 0.82403861 0.75857904 0.67976189 0.77320399]
avg score: 0.7613083688845369
-----test_MSE (0.0 is best)
scores:   [-54.02552957 -44.2356364  -57.58504619 -47.55941503 -54.9157
0954]
avg score: -51.66426734736389
-----test_R2 (1.0 is best)
scores:   [0.7358449  0.8238177  0.75544849 0.67836124 0.76208865]
avg score: 0.7511121965432139


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

K-Nearest Neighbors reaches near maximum accuracy at 5 neighbors and ri
sing marginally until 9 neighbors

----------K-Nearest Neighbors, 3-neighbors scores:
-----test_FVE (1.0 is best)
scores:   [0.67132017 0.5897223  0.542423   0.50447341 0.42788093]
avg score: 0.5471639600628959
-----test_MSE (0.0 is best)
scores:   [ -69.27011494 -104.1302682  -108.79310345  -73.36781609 -14
2.06822612]
avg score: -99.52590576056998
-----test_R2 (1.0 is best)
scores:   [0.66130726 0.58526854 0.53797871 0.50382205 0.38451777]
avg score: 0.5345788656771712

----------K-Nearest Neighbors, 5-neighbors scores:
-----test_FVE (1.0 is best)
scores:   [0.71106917 0.58881426 0.65815358 0.55178016 0.46796877]
avg score: 0.5955571874020688
-----test_MSE (0.0 is best)
scores:   [ -61.32206897 -103.30689655  -80.54551724  -66.42827586 -13
4.19929825]
```

Much like the previous problem, K-nearest neighbors performed by far the worse and random forests performed very well even at low tree counts.