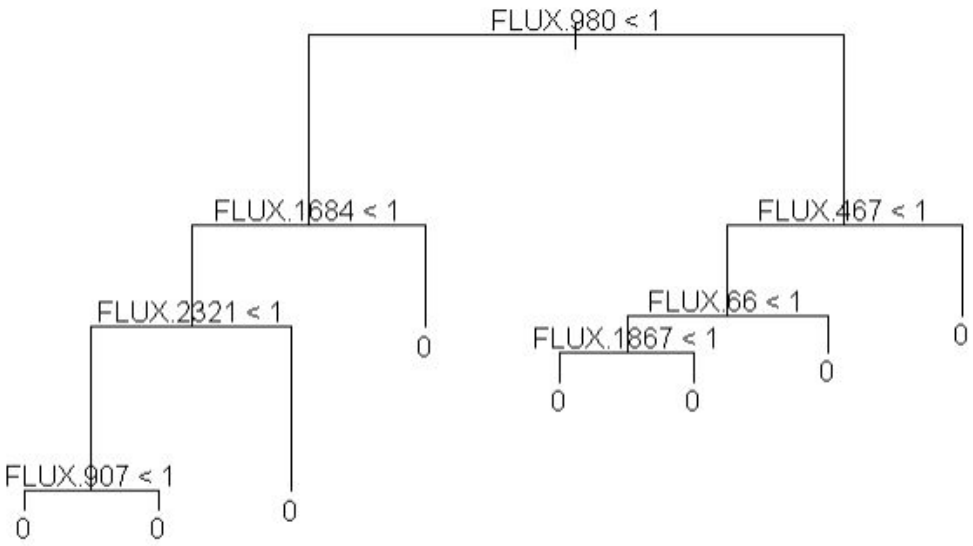
The project I did involved taking data from the following LINK, and testing it to see if it is possible to predict LABEL based off of the other variables. About the data, there are 2 datasets, exoTrain and exoTest. Both measure the same thing, different stars with potential of exoplanets at different points in time. There are 3198 variables, LABEL and Flux.0001 to Flux.2197. LABEL is either 1 or 0, representing yes to an exoplanet (1), or no to an exoplanet (0). Flux.XXXX are the different points in time. A flux signifies the light intensity of a star. At any given point in time a flux is the same across all stars, so Flux.2888 is the same time for all stars. exoTest is separated from exoTrain, but the data is the same for both, with the only difference being that exoTest has only 570 stars while exoTrain has 5087 stars. For a reasonable runtime, exoTrain will be omitted from my code.

To predict an exoplanet, I used trees to divide entries based on values and patterns of different Fluxes. However, due to large disparity of the values of fluxes, with ranges from negative millions to positive millions, there needs to be some modification to the data. Therefore, the problem can be split into 2 parts: the restructuring of the data to be more manageable, and then the computation of the trees.

To restructure the data, I explored 2 methods. Firstly, is the idea of thresholding the data to the 95% range. This is because almost all fluxes have 95% of their points within a range of 2000 to 3000, centred at or near 0. That is very good, compared to ranges from positive to negative millions. The second is the idea of condensing every flux to be within -1 to 1. This was done by forcing every flux through the equation x = (2 / (1+e^-x)) + 1. Both of these methods managed to make the data much more manageable, while preserving really high accuracy. Thresholding accuracy was about 94%, while using the equation x = (2 / (1+e^-x)) + 1 had an accuracy of about 96%, so I used the equation (2 / (1+e^-x)) + 1 for the tree.

The trees are where the actual discovery comes. For the trees, I set LABEL as the terminal node of the tree. I got pretty good results, with the misclassification error to be 0.8%, and the accuracy of the to be 99.1%. Unfortunately, both attempts at cross validation did not work. The built in functions did not manage to compile, and so were omitted. The manual attempts output the same results every time.

```
Classification tree:
tree(formula = High ~ . - LABEL, data = planetTree, control = tree.control(3199,
    mincut = 10))
Variables actually used in tree construction:
[1] "FLUX.980"  "FLUX.1684" "FLUX.2321" "FLUX.907"  "FLUX.467"  "FLUX.66"   "FLUX.1867"
Number of terminal nodes:  8
Residual mean deviance:  0.04751 = 26.7 / 562
Misclassification error rate: 0.008772 = 5 / 570
```




```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 570 57.320 0 ( 0.991228 0.008772 )
   2) FLUX.980 < 1 102 33.750 0 ( 0.960784 0.039216 )
     4) FLUX.1684 < 1 56 28.820 0 ( 0.928571 0.071429 )
       8) FLUX.2321 < 1 22 20.860 0 ( 0.818182 0.181818 )
        16) FLUX.907 < 1 12 13.500 0 ( 0.750000 0.250000 ) *
        17) FLUX.907 > 1 10  6.502 0 ( 0.900000 0.100000 ) *
       9) FLUX.2321 > 1 34  0.000 0 ( 1.000000 0.000000 ) *
     5) FLUX.1684 > 1 46  0.000 0 ( 1.000000 0.000000 ) *
   3) FLUX.980 > 1 468 14.290 0 ( 0.997863 0.002137 )
     6) FLUX.467 < 1 53  9.922 0 ( 0.981132 0.018868 )
      12) FLUX.66 < 1 22  8.136 0 ( 0.954545 0.045455 )
        24) FLUX.1867 < 1 11  6.702 0 ( 0.909091 0.090909 ) *
        25) FLUX.1867 > 1 11  0.000 0 ( 1.000000 0.000000 ) *
      13) FLUX.66 > 1 31  0.000 0 ( 1.000000 0.000000 ) *
     7) FLUX.467 > 1 415  0.000 0 ( 1.000000 0.000000 ) *
```

Link in full detail:
https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data