After data is loaded few preprocessing steps needs to performed on data:

* Discretize - Splits Numerical data columns into labels for each bin with an equal amount of elements in each bin. And it will be represented as an integer.
* Decode - Turns integer categories into strings with labels. Basically converting the int column name to a string column name. And then modify the column row data to a combination of column row data + new column name
* Gen\_freq - Return a dict of label(column+row value) and its freq
* Convert\_dict - converts above dictionary with frequency counts to nodes with name and frequency count

**The Tree class is initialized with output of Covert\_dict func and a Root Node**

* Generate - generate the tree from the original database. We create the left node for root only once. The remaining all nodes are created for right only. If while checking the nodes are not matching to the left value then we start checking right and if we still don’t find it we will then create a new right node.
* Generatesub - generate sub tree based on the path with the support of the path

**FPGrowth:**

* If there is one path(i.e root just only have left node), print all combinations of all elements. This can be found by only traversing the left nodes which are not having any corresponding right nodes.
* After you reach the end you can check if there are any nodes left behind if not then : print all combinations of all elements
* Else: Then use the dictionary to look up
  + - The recursive format for generating freq pattern  
      if paths != []:

btree.generatesub(paths, supports)

condpatbase[key] = (paths, supports)

a.append(key)

FP\_growth(btree, a, min\_sup)