Advanced Tutorial on Implementing CPGrowth Algorithm

In this tutorial, we explain how the Correlated Pattern Growth (CPGrowth) algorithm can be implemented by varying the minimum support values

Step 1: Import the CPGrowth algorithm and pandas data frame

In [1]:
```
from PAMI.correlatedPattern.basic import CPGrowth  as alg
import pandas as pd
```

Step 2: Specify the following input parameters

In [2]:
```
inputFile = 'transactional_T10I4D100K.csv'
seperator='¥t'
minAllConfCount=0.1
minimumSupportCountList = [100, 150, 200, 250, 300]
#minimumSupport can also specified between 0 to 1. E.g., minSupList = [0.005, 0.006,

result = pd.DataFrame(columns=['algorithm', 'minSup',"minAllConf" , 'patterns', 'rur
#initialize a data frame to store the results of CPGrowth algorithm
```

Step 3: Execute the CPGrowth algorithm using a for loop

In [3]:
```
algorithm = 'CPGrowth'  #specify the algorithm name
for minSupCount in minimumSupportCountList:
    obj = alg.CPGrowth('transactional_T10I4D100K.csv', minSup=minSupCount,minAllConf
    obj.startMine()
    #store the results in the data frame
    result.loc[result.shape[0]] = [algorithm, minSupCount,minAllConfCount, len(obj.g
```

```
IOPub data rate exceeded.
The Jupyter server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--ServerApp.iopub_data_rate_limit`.

Current values:
ServerApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
ServerApp.rate_limit_window=3.0 (secs)
```

```
['32', '239', '372', '419', '448', '510', '540', '581', '674', '752', '802', '844',
'887', '922']
Correlated Frequent patterns were generated successfully using CorrelatedPatternGrow
th algorithm
```

In [4]:
```
print(result)
```

```
  algorithm  minSup  minAllConf  patterns     runtime     memory
0  CPGrowth     100         0.1      5758  11.793282  411676672
1  CPGrowth     150         0.1     15302  10.779621  490831872
2  CPGrowth     200         0.1     22339  11.608838  498159616
3  CPGrowth     250         0.1     28538  11.813459  504766464
4  CPGrowth     300         0.1     33076  12.897403  513916928
```

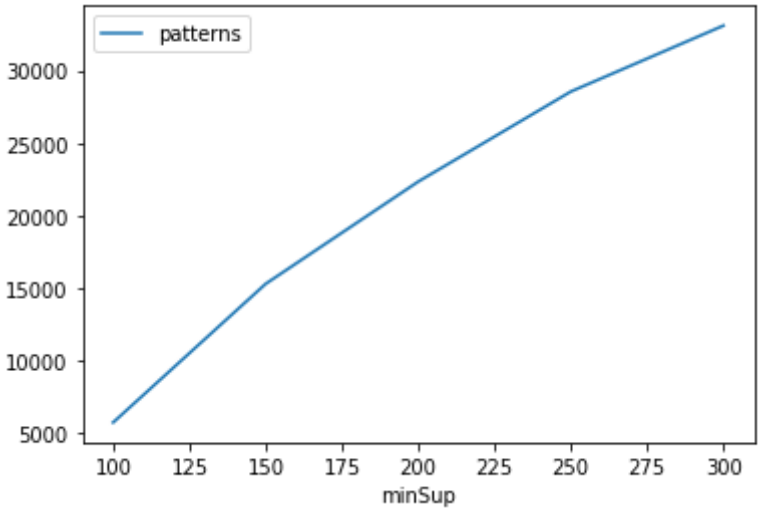
Step 5: Visualizing the results

Step 5.1 Importing the plot library

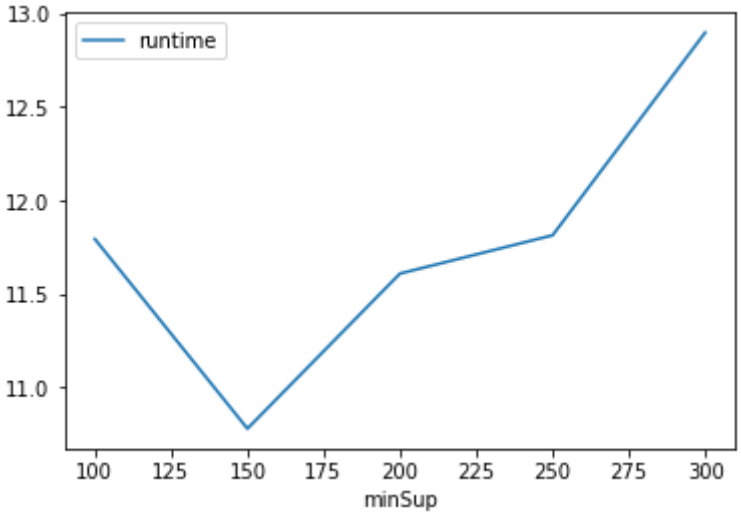
In [5]:
```
from PAMI.extras.graph import plotLineGraphsFromDataFrame as plt
```

Step 5.2. Plotting the number of patterns

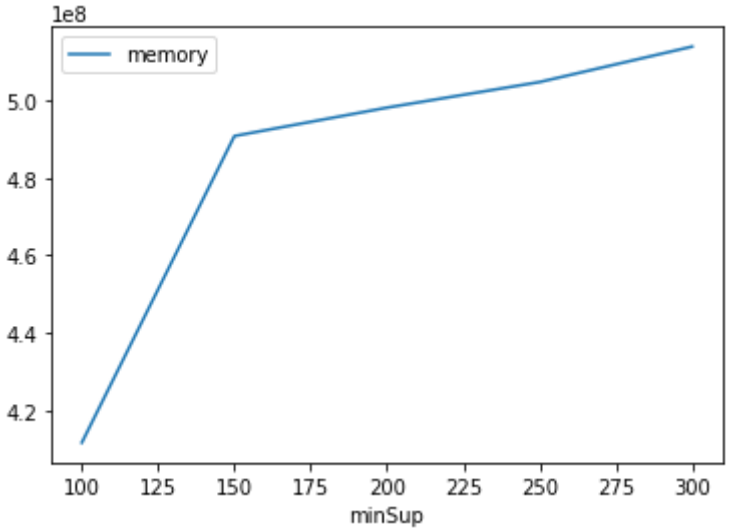
In [6]:
```
ab = plt.plotGraphsFromDataFrame(result)
ab.plotGraphsFromDataFrame() #drawPlots()
```




```
Graph for No Of Patterns is successfully generated!
```




```
Graph for Runtime taken is successfully generated!
```




```
Graph for memory consumption is successfully generated!
```

Step 6: Saving the results as latex files

In [7]:
```
from PAMI.extras.graph import generateLatexFileFromDataFrame as gdf
gdf.generateLatexCode(result)
```

```
Latex files generated successfully
```