

# How to execute the code:

**We have created an abstract class named “PeriodicPatterns” inside the “abstractPeriodicPatterns.py” python file. Therefore, every program has to import this file and needs to extend the abstract class as follows:**

- *from traditional.abstractClass.abstractPeriodicFrequentPatterns import \**
- *class Eclatpfp(frequentPatterns):*
  - *Complete code along with the implementation of the given abstract .*

## **1. Periodic Frequent Pattern Mining (PFPM) Process:**

- 1.1.Import our package and initialize the method called '**EclatPFP**' using the input file path/input file and minimum support (It has to be given in terms of count of total number of transactions in the input database/file).
- 1.2.Then call the method '**startMine**' using the following command

```
import eclatpfp as Myap  
fp= Myap.Eclatpfp()  
fp.iFile = “file name”  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using efclatpfp algorithm.

For example:

If we execute the following command:

```
import eclatpfp as Myap
```

```
fp= Myap.Eclatpfp()  
fp.iFile = “file name”
```

**fp.minSup = minimum support**  
**fp.maxPer = maximum period**  
**fp.startMine()**

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.

2. To get the periodic-frequent patterns along with their support count:

2.1. Complete the PFP Process mentioned in **(1)**

2.2. Then call the method '**getPeriodicPatterns**' using the following command:

```
import eclatpfp as Myap  
fp= Myap.Eclatpfp()  
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
variable = fp.getPeriodicPatterns()
```

output is displayed as follows:

- periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- All the Periodic-Frequent patterns will be stored in a dictionary, with patterns as keys and support count, periodicity as value and returned to the called function.

For example:

If we execute the following command:

```
import eclatpfp as Myap  
  
fp= Myap.Eclatpfp()  
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
periodicPatterns = fp.getPeriodicPatterns()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.

- All the Periodic-Frequent patterns will be stored in a dictionary, with patterns as keys and support count, periodicity as value and assigned to the variable called '**periodicPatterns.**'
3. To get the periodic-frequent patterns along with their support count in a file:
    - 3.1. Complete the PFPM Process mentioned in **(1)**
    - 3.2. Then call the method '**storePatternsInFile**' using the following command:

```
import eclatpfp as Myap
fp= Myap.Eclatpfp()
fp.iFile = "file name"
fp.minSup = minimum support
fp.maxPer = maximum period
fp.startMine()
fp.storePatternsInFile("output file")
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- All the Periodic-Frequent patterns will be stored in a file named as "output file"

For example:

If we execute the following command:

```
import eclatpfp as Myap

fp= Myap.Eclatpfp()
fp.iFile = "file name"
fp.minSup = minimum support
fp.maxPer = maximum period
fp.startMine()
fp.storePatternsInFile("sampleoutput")
```

output is displayed as follows:

- periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- All the Periodic-Frequent patterns will be stored in a file named as 'sampleoutput.'

4. To get the periodic-frequent patterns along with their support count in a DataFrame:
  - 4.1. Complete the PFPM Process mentioned in **(1)**

4.2. Then call the method **'getPatternsInDataFrame'** using the following command:

```
import eclatpfp as Myap  
fp= Myap.Eclatpfp()  
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
variable =fp.getPatternsInDataFrame()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- All the Periodic-Frequent patterns will be stored in a data frame, their columns named as 'Patterns' and 'Support, periodicity' and returned to the called function.

For example:

If we execute the following command:

```
import eclatpfp as Myap  
  
fp= Myap.Eclatpfp()  
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
dataFrame= fp.getPatternsInDataFrame()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- All the Periodic-Frequent patterns will be stored in a data frame, their columns named as 'Patterns' and 'Support,periodicity' and stored in a variable called 'dataFrame.'

5. If we want to know the amount of USS memory consumed by the eclatpfp algorithm:

5.1. Complete the PFP Process mentioned in **(1)**

5.2. Then call the method **'getMemoryUSS'** using the following command:

```
import eclatpfp as Myap  
fp= Myap.Eclatpfp()
```

```
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
variable = fp.getMemoryUSS()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- Total amount of USS memory consumed by the program will be computed and returned to the called function.

For example:

If we execute the following command:

```
import eclatpfp as Myap  
  
fp= Myap.Eclatpfp()  
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
memoryUSS = fp.getMemoryUSS()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- Total amount of USS memory consumed by the program will be computed and returned to the variable called '**memoryUSS.**'

6. If we want to know the amount of RSS memory consumed by the eclatpfp algorithm:

6.1. Complete the PFPM Process mentioned in **(1)**

6.2. Then call the method '**getMemoryRSS**' using the following command:

```
import eclatpfp as Myap  
  
fp= Myap.Eclatpfp()  
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
variable = fp.getMemoryRSS()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- Total amount of RSS memory consumed by the program will be computed and returned to the called function.

For example:

If we execute the following command:

```
import eclatpfp as Myap

fp= Myap.Eclatpfp()
fp.iFile = "file name"
fp.minSup = minimum support
fp.maxPer = maximum period
fp.startMine()
memoryRSS = fp.getMemoryRSS()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- Total amount of RSS memory consumed by the program will be computed and returned to the variable called ' **memoryRSS.**'

7. If we want to know the runtime taken by the eclatpfp algorithm created by us:

7.1.Complete the PFPM Process mentioned in **(1)**

7.2.Then call the method '**getRuntime**' using the following command:

```
import eclatpfp as Myap
fp= Myap.Eclatpfp()
fp.iFile = "file name"
fp.minSup = minimum support
fp.maxPer = maximum period
fp.startMine()
variable = fp.getRuntime()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- Total runtime taken by the program in seconds will be computed and returned to the called function.

For example:

If we execute the following command:

```
import eclatpfp as Myap
```

```
fp= Myap.Eclatpfp()  
fp.iFile = "file name"  
fp.minSup = minimum support  
fp.maxPer = maximum period  
fp.startMine()  
run = fp.getRuntime()
```

output is displayed as follows:

- Periodic-Frequent patterns were generated successfully using eclatpfp algorithm.
- Total runtime taken by the program in seconds will be computed and returned to the variable called 'run.'