## Dataflow programming with mlr3pipelines::CHEAT SHEET

### Introduction
Combine ML operations to flexible pipelines and processing graphs, which can be configured, trained, resampled, tuned as any regular learner. The main purpose of a Graph is to build combined preprocessing and model fitting pipelines that can be used as a Learner.

### Graph
Connects PipeOps with edges to control data flow during training and prediction. Input is sent to sources (no out-edges). Important methods and slots:
- `display(print)`, `plot(html = TRUE)`
- Accessing PipeOps: `gr$pipeops` Named list of all contained POs.

### Graph Construction
The `%>%` operator takes either a PipeOp or a Graph on each of its sides and connects all left-hand outputs to the right-hand inputs. For full control, connect PipeOps explicitly:

```r
g = Graph$new()
g$add_pipeop(po("pca"))
g$add_edge("pca", "classif.rpart")
```

### Linear Graphs
Concatenates PipeOps with `%>%`. Usage of AutoTuner is identical.

### Popular PipeOps
<table>
<thead>
<tr>
<th>Class</th>
<th>Key</th>
<th>Operation</th>
<th>Repair Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>PipeOpRemoveConstants</td>
<td>removeconstructs</td>
<td>&quot;scale&quot; Scale Features</td>
<td></td>
</tr>
<tr>
<td>PipeOpScale</td>
<td>&quot;scale&quot;</td>
<td>Impute Mean</td>
<td></td>
</tr>
<tr>
<td>PipeOpImputeMean</td>
<td>&quot;impute&quot;</td>
<td>Feature Filter</td>
<td></td>
</tr>
<tr>
<td>PipeOpPCA</td>
<td>&quot;pca&quot;</td>
<td>PCA</td>
<td></td>
</tr>
<tr>
<td>PipeOpSelect</td>
<td>&quot;select&quot;</td>
<td>Factor Encoding</td>
<td></td>
</tr>
<tr>
<td>PipeOpGreplicate</td>
<td>&quot;greplicate&quot;</td>
<td>Restrict Columns</td>
<td></td>
</tr>
<tr>
<td>PipeOpClassBalancing</td>
<td>&quot;classbalancing&quot;</td>
<td>Imbalanced Data</td>
<td></td>
</tr>
<tr>
<td>PipeOpLearn</td>
<td>&quot;learner&quot;</td>
<td>Use Learner</td>
<td></td>
</tr>
<tr>
<td>PipeOpLearnCV</td>
<td>&quot;learner.cv&quot;</td>
<td>Crossval Learner</td>
<td></td>
</tr>
<tr>
<td>PipeOpMutate</td>
<td>&quot;mutate&quot;</td>
<td>Feature Engineering</td>
<td></td>
</tr>
<tr>
<td>PipeOpChurn</td>
<td>&quot;churn&quot;</td>
<td>Split Data</td>
<td></td>
</tr>
<tr>
<td>PipeOpSubsample</td>
<td>&quot;subsample&quot;</td>
<td>Subsample Rows</td>
<td></td>
</tr>
<tr>
<td>PipeOpFeatureUnion</td>
<td>&quot;featureunion&quot;</td>
<td>Combine Features</td>
<td></td>
</tr>
<tr>
<td>PipeOpFixFactors</td>
<td>&quot;factors&quot;</td>
<td>Handle Unknown Levels</td>
<td></td>
</tr>
<tr>
<td>PipeOpNOP</td>
<td>&quot;nop&quot;</td>
<td>Do Nothing</td>
<td></td>
</tr>
</tbody>
</table>

### Hyperparameters
For POs: Exactly as in a Learner.

### Feature Engineering
PipeOpMutate adds new features. This works by providing expressions in a list.

```r
g = po("scale")
muts = list("Sepal.Sum = Sepal.Length + Sepal.Width")
g = po("mutate", params = list(muts = muts))
```

### Nonlinear Graphs
`gunion()` arranges PipeOps or Graphs next to each other in a disjoint graph union.

### Tuning
Can jointly tune any Pipeline.

```r
g = GraphLearner$new()
g = g$add_pipeop(po("pca"))
g = g$add_pipeop(po("classif.rpart"))
```

### Pipelines
PipeOpFeatureUnion aggregates features from all input tasks into a single Task.

```r
g = GraphLearner$new()
g = g$add_pipeop(po("classif.rpart"))
g = g$add_pipeop(po("classif.rpart"))
```

### Branching
Controls the path execution. Only one branch can be active. Which one is controlled by a hyperparameter. Unbranching ends the forking.

```r
g = GraphLearner$new()
g = g$add_pipeop(po("classif.rpart"))
g = g$add_pipeop(po("classif.rpart"))
```

Tuning the branching selection enables powerful model selection.