ML in production

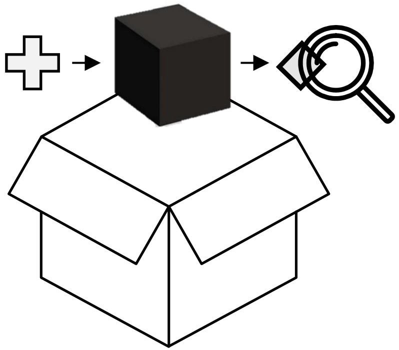
Automation of ML pipelines with Luigi

Eliana Pastor

Data Base and Data Mining Group of Politecnico di Torino

Eliana Pastor

Post-doc



Main research topics
- Explainable AI
- Fairness in ML
- Exploratory data analysis
- Predictive Maintenance
- Industrial ML

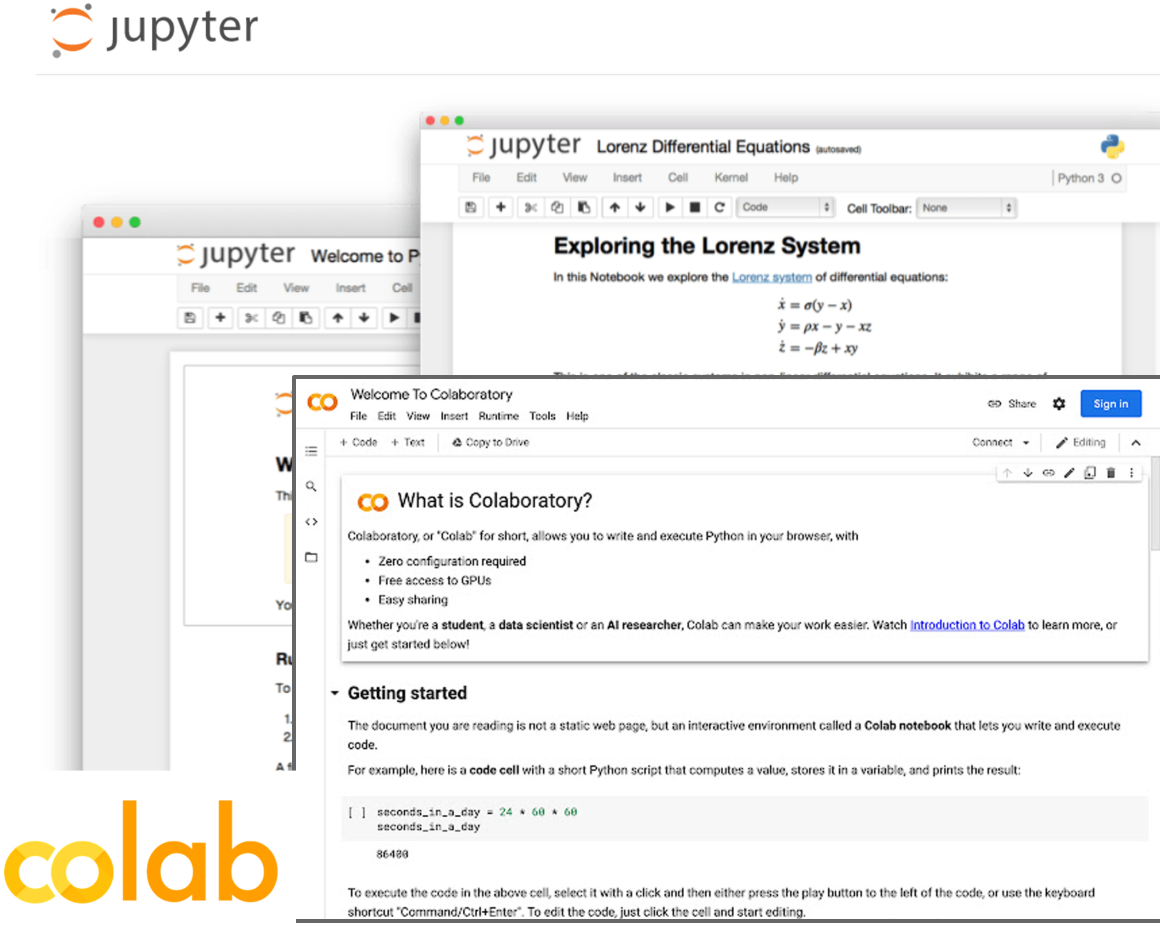
ML pipeline

Data ingestion	Data selection	Data preprocessing	Data transformation	Modeling	Model deployment
Data	Selected data	Processed data	Transformed data	Model	Results (predictions)

ML pipeline

Notebooks
E.g. jupyter notebooks

- Prototyping
- Exploratory and evaluation phase
- Share results and analysis



ML pipeline

Scripts

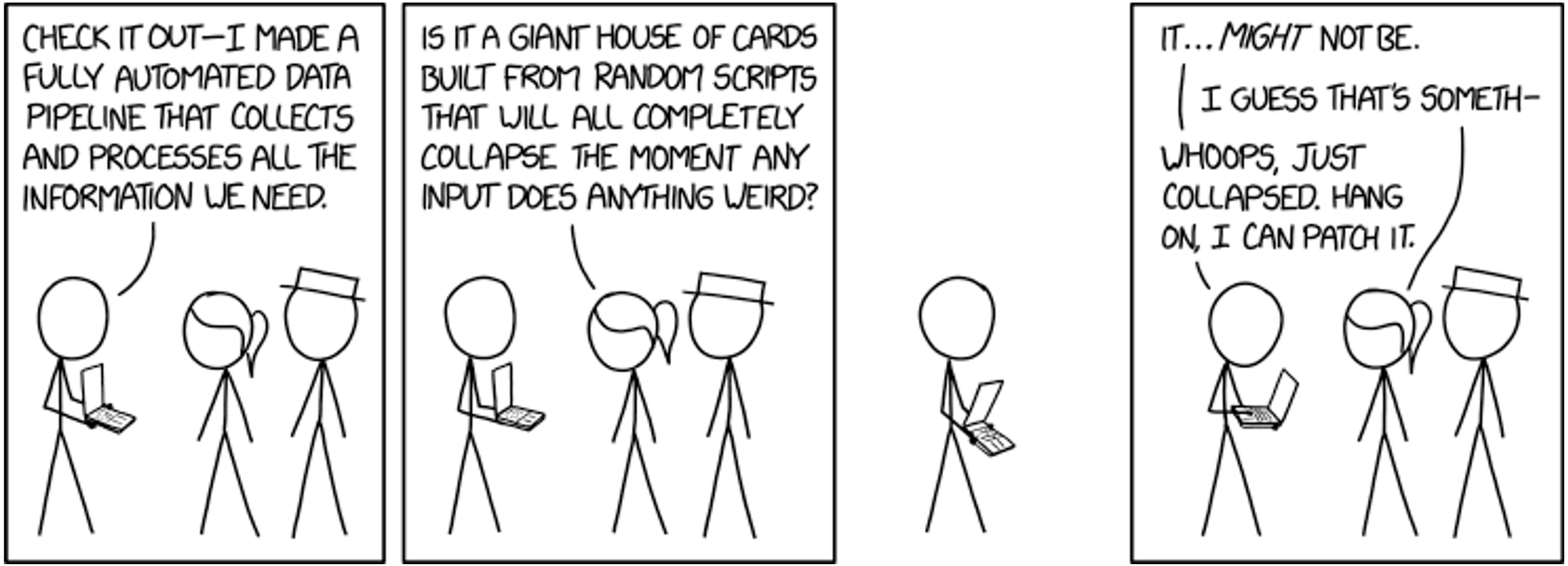
- Multiple files

- Sequential run

- From data import to model prediction

```
data=load_data(data_location)

data_selected=select_data(data)

processed_data=preprocess_data(data_selected)

transformed_data=transform_data(processed_data)

model=modeling(processed_data)
```

ML pipeline



https://xkcd.com/2054

ML in production… or not?

Why do 87% of data science projects never make it into production?

VB Staff July 19, 2019 4:10 AM **AI**

October 16, 2018 | Contributor: Kasey Panetta

Through 2020, 80% of AI projects will remain alchemy, run by wizards whose talents will not scale in the organization.

In the past five years, the increasing popularity and hype surrounding AI techniques have led to an increase in projects across organizations. However, the overwhelming hype has also led to unreasonable expectations from the business. Further, change is outpacing the production of competent professionals, which means that AI is more an art form than a science. The lack of a common language among all parties remains as a barrier to scalability, as does how specific and narrow most AI skill sets remain. Combining new skills with AI-based automation will unlock scale potential.

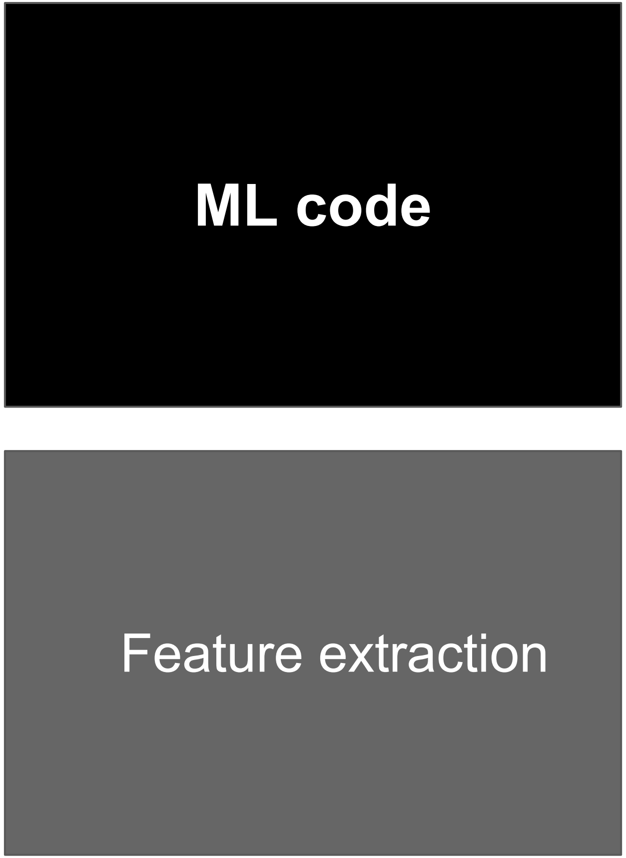
https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/
https://www.gartner.com/smarterwithgartner/gartner-top-strategic-predictions-for-2019-and-beyond/

ML systems

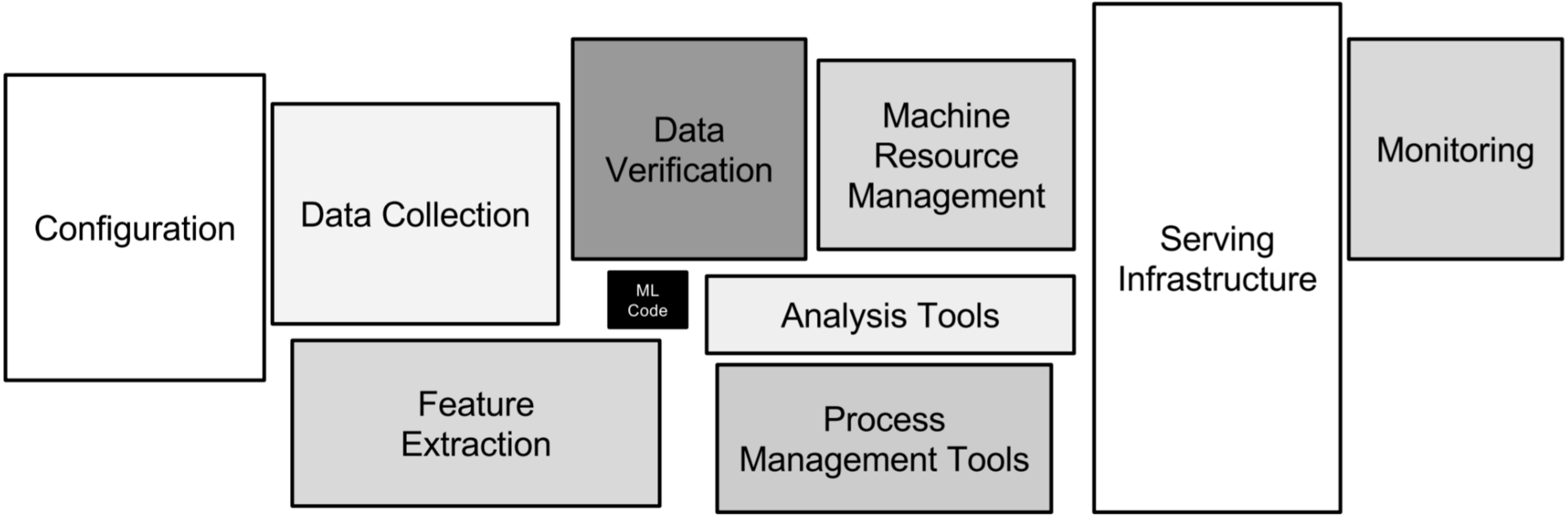
- Machine learning code
  - Modeling

- Feature extraction and engineering

….



Hidden technical debt in ML systems



Sculley, David, et al. "Hidden technical debt in machine learning systems." *Advances in neural information processing systems*. 2015. NIPS'15

Technical debt

Concept of software development

The implied cost of additional rework caused by choosing an easy (and limited) solution instead of using a better approach that would take longer and is harder to implement

In software development, technical debt may be handled by:

- refactoring code
- improving unit tests
- deleting dead code
- reducing dependencies
- improving documentation

Technical debt in ML pipelines

Not only code level...

- maintenance problems of traditional code

… at the system level

Data

Directly influences the behavior of ML systems

Technical debt in ML pipelines

Entanglement

- Machine learning systems mix input together, entangling them and the isolation of improvement and components is difficult

  CACE issue: Changing Anything Changes Everything

  e.g. If an input feature changes, then the importance, weights or use of the remaining features may all change as well (or not).

  CACE applies to input signals, hyper-parameters, learning settings, sampling methods, convergence thresholds, data selection..

Technical debt in ML pipelines

Entanglement

Mitigation strategy:

- Isolate models
- **Monitoring**
  - **detecting changes in prediction behavior as they occur.**
  - ML systems must be designed so that feature engineering and selection changes are easily tracked.

Technical debt in ML pipelines

Unstable Data Dependencies

- Input features may be produced by other systems
- Problem: some data inputs are unstable, changing behavior over time.
    - E.g. they are output of another ML model that updates over time
    - E.g. they are generated by a model that was refactored or reconfigured (calibration)

Technical debt in ML pipelines

Unstable Data Dependencies

Mitigation strategy

- Monitoring:
  - track the changes
- Versioning
  - Create a versioned copy of a given signal
  - Problems
    - “model staleness”, i.e. predictive power of a ML model decreasing over time
    - the cost to maintain multiple versions of the same signal over time

Technical debt in ML pipelines

Underutilized Data Dependencies

- For code, we may have packages or functions mostly unneeded
- For data dependencies, we may have **data inputs that provided just a little incremental modeling benefit (or none if unneeded)**
    - also unneeded over time

Problem: they represent an unnecessarily vulnerability to change of the ML system

Technical debt in ML pipelines

Underutilized Data Dependencies

Examples

- **ε-Features**. → Features that increase model performance of just a very small ε but there is a high complexity overhead to include and maintaining them
- **Correlated Features**. → ML methods may have difficulty detecting the correlation and credit the two correlated features equally or only the non-causal one is used as input. We may have problem if later the behavior changes and correlations change.
- **Legacy Features**. → A feature F that is included in a model early in its development but over time, F is made redundant by new features but this goes undetected.
- **Bundled Features**. → A group of features is evaluated to be beneficial. All of features in the groups are added as input without considering the one that actually add values.

Technical debt in ML pipelines

Underutilized Data Dependencies

- Mitigation strategy
  - Monitoring:
    - track the changes, especially changes in correlations and redundancy features
    - Detect them
      - Exhaustive leave-one-feature-out evaluations that should be performed regularly to identify and remove unnecessary feature

Technical debt in ML pipelines

Configuration

Configuration Debt

In ML systems we have a wide range of configurable options:

- as which features are used, how data is selected, a wide variety of algorithm-specific learning settings, potential pre- or post-processing, verification methods, etc

Problem: mistakes in configuration can be costly, leading to serious loss of time, waste of computing resources, or production issues

Technical debt in ML pipelines

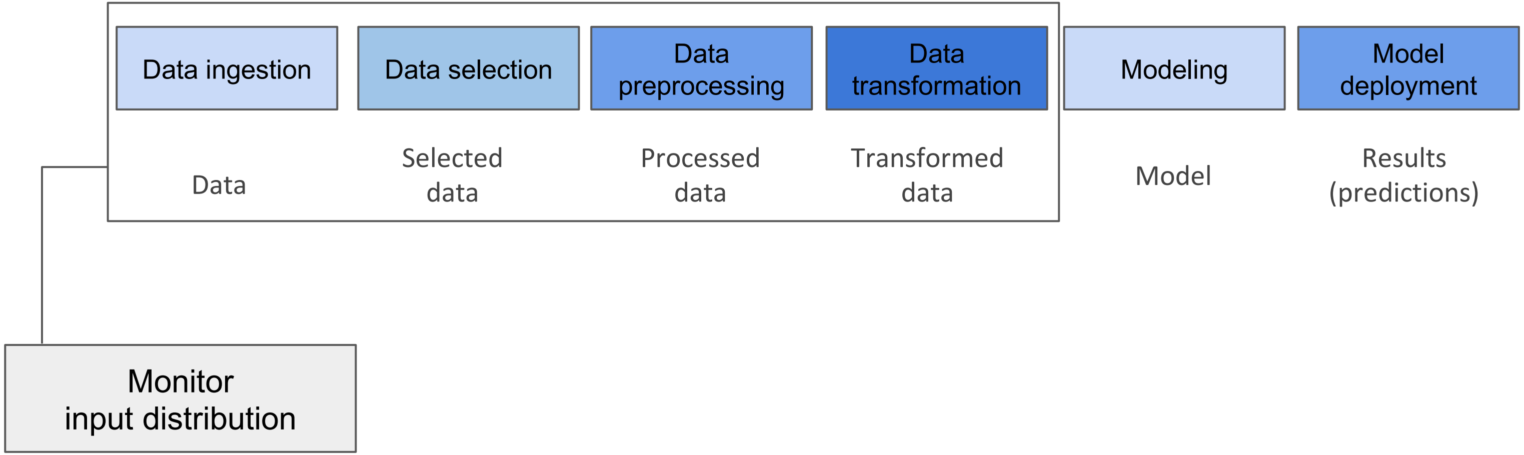
Configuration Debt

Example of principles of good configuration systems*:

- It should be easy to specify a configuration as a small change from a previous configuration.
- It should be hard to make manual errors, omissions, or oversights.
- It should be easy to see, visually, the difference in configuration between two models.
- It should be easy to automatically assert and verify basic facts about the configuration: number of features used, transitive closure of data dependencies, etc.
- It should be possible to detect unused or redundant settings.
- Configurations should undergo a full code review and be checked into a repository.

* Sculley, David, et al. "Hidden technical debt in machine learning systems." *Advances in neural information processing systems*. 2015. NIPS'15

ML pipeline



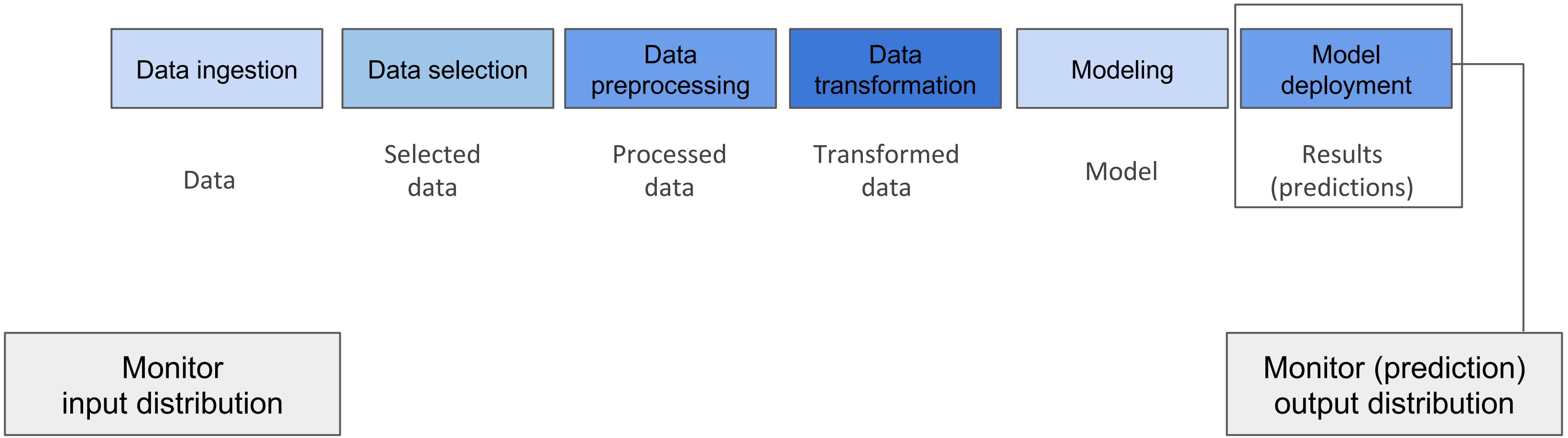
Data dependencies

- Changing data, Underutilized data, Unstable data

Monitor the distribution of **inputs** to detect changes

- If the model is not able to adapt to the change → effect on predictions

ML pipeline

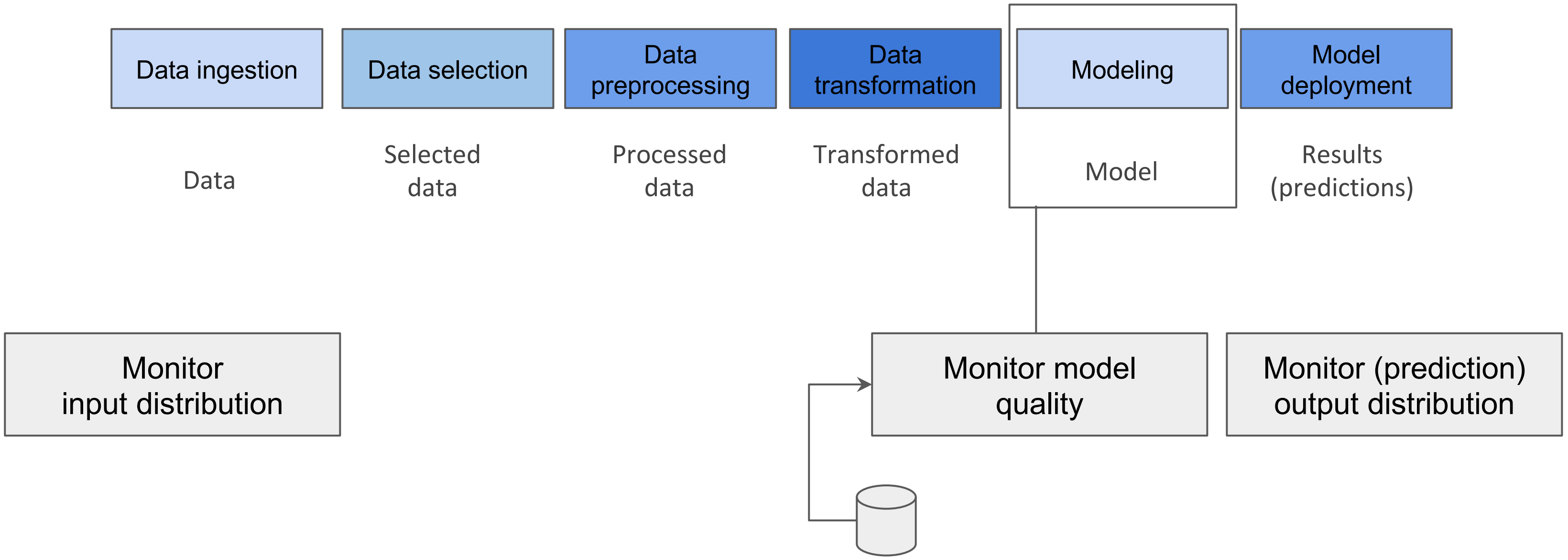


Monitor the distribution of **outputs** to detect changes

- Observe effect of change and data dependency on the output

e.g. fraud detection problem, training is highly imbalanced (99% transactions are legal and 1% are fraud). But over time, the system labels 20% of transactions as fraudulent.

ML pipeline



Monitor the quality of the model

- Monitor model performance on new test data to test model quality

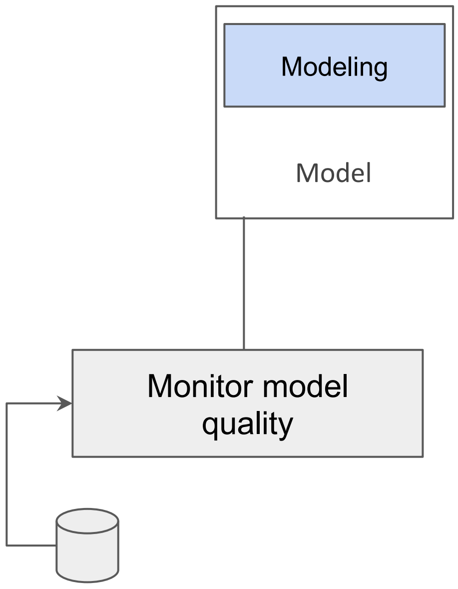
Model staleness - Model drifting

Data can change over time

A model that was initially working could later degrade due to a **data drift** or **concept drift**.

Concept drift refers to the change in the relationships between input and output data in the underlying problem over time.

Mitigation approach
- Monitoring the model performance



Model staleness - Model drifting

Monitoring the model performance

If the model falls below an acceptable performance threshold

- Model retraining
- Deploying the new model

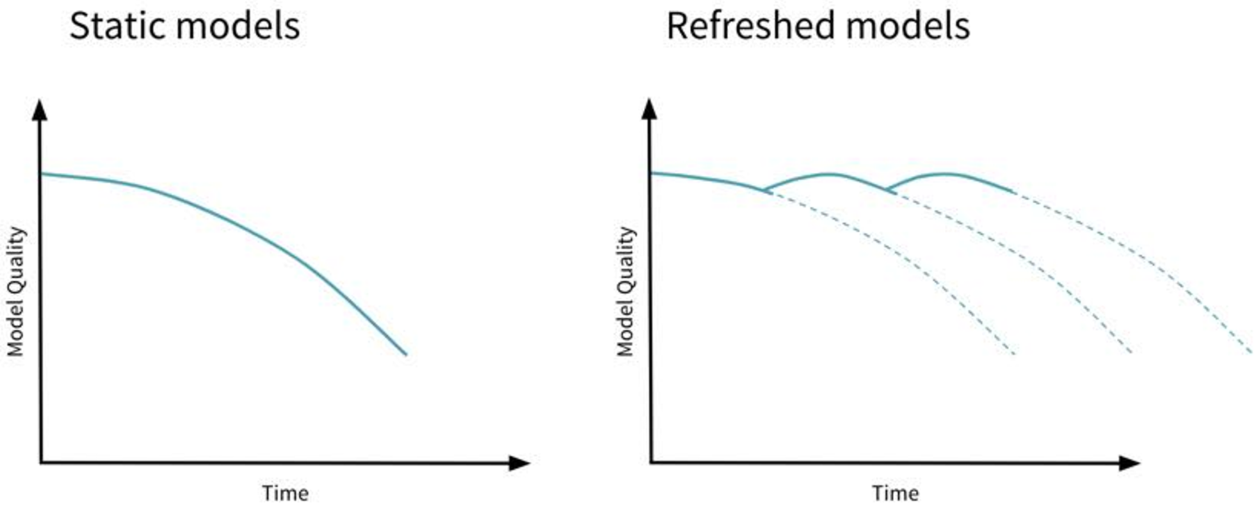


Image credit https://databricks.com/blog/2019/09/18/productionizing-machine-learning-from-deployment-to-drift-detection.html

Versioning

Versioning: store and assign an unique identifier

- Code
- Model (and its hyperparameters)
- Data
  - Data
  - Features

Pachyderm
version-controlled data science

Data Version Control

Goal

- Reproducibility
- Failure tolerance
- Version comparison (e.g. old vs new model)
- Understanding how data and results change over time

Data ingestion

Data selection

Data preprocessing

Data transformation

Modeling

Model deployment

Data

Selected data

Processed data

Transformed data

Model

Results (predictions)

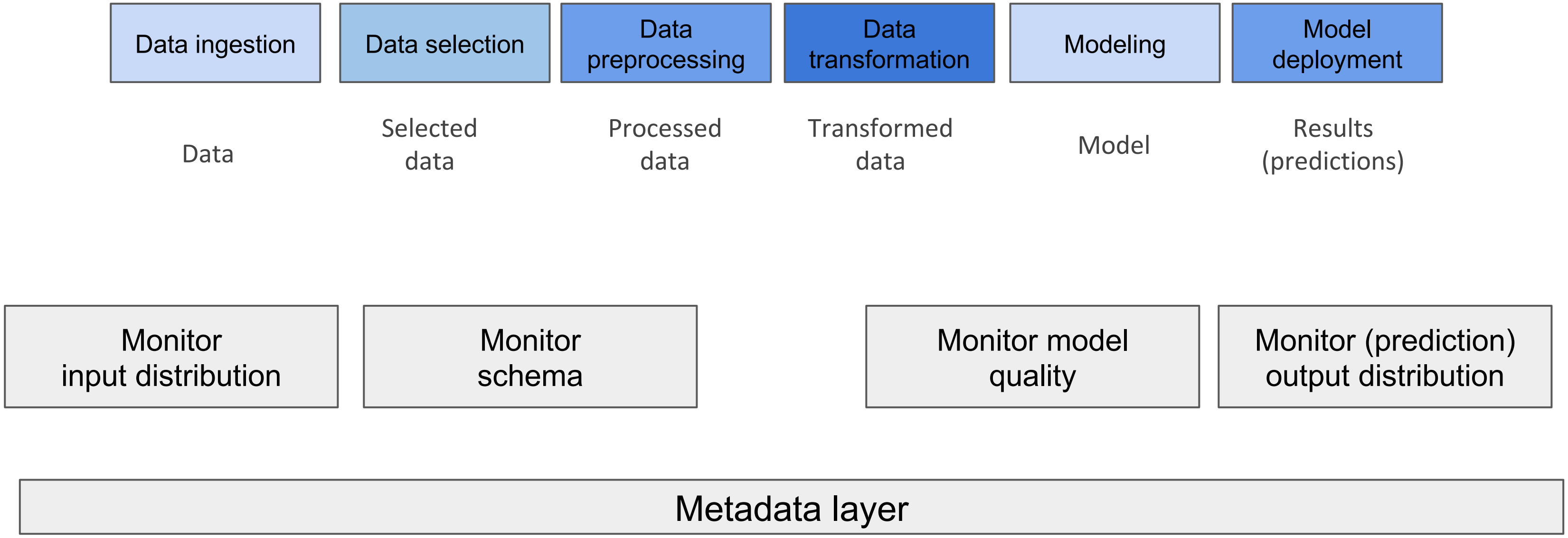
Monitor input distribution

Monitor schema

Monitor model quality

Monitor (prediction) output distribution

Metadata layer

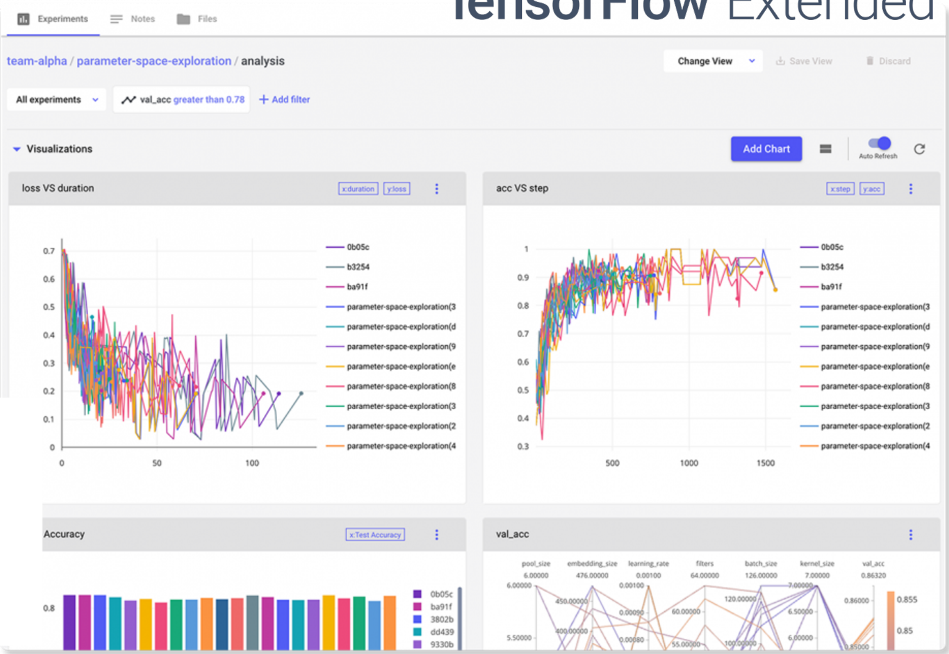


Metadata layer

Keep track

- Results and their properties and location
- Configuration between components → execution records
- Interaction among components → flow through the pipeline

TensorFlow Extended



Metadata tracking tools

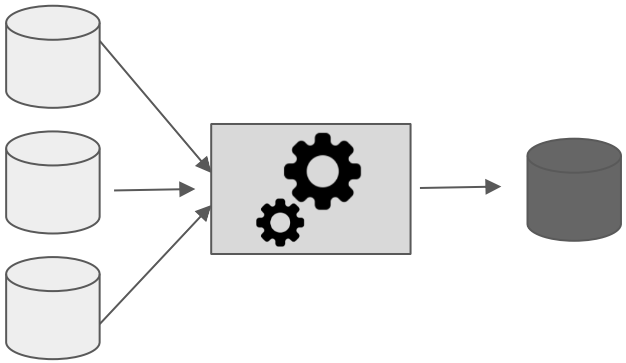
- ML Metadata library in TensorFlow Extended
- Comet ML

comet

Data sources and features

Data ingestion

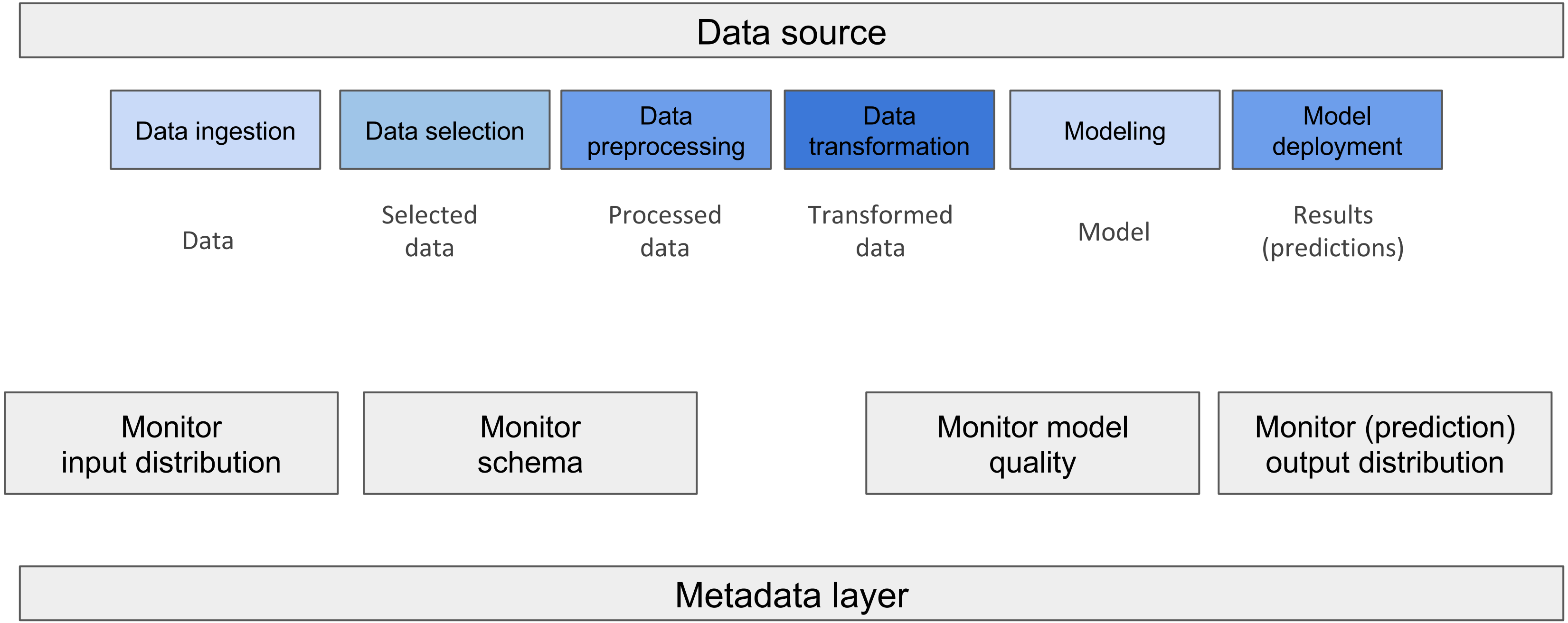
Data



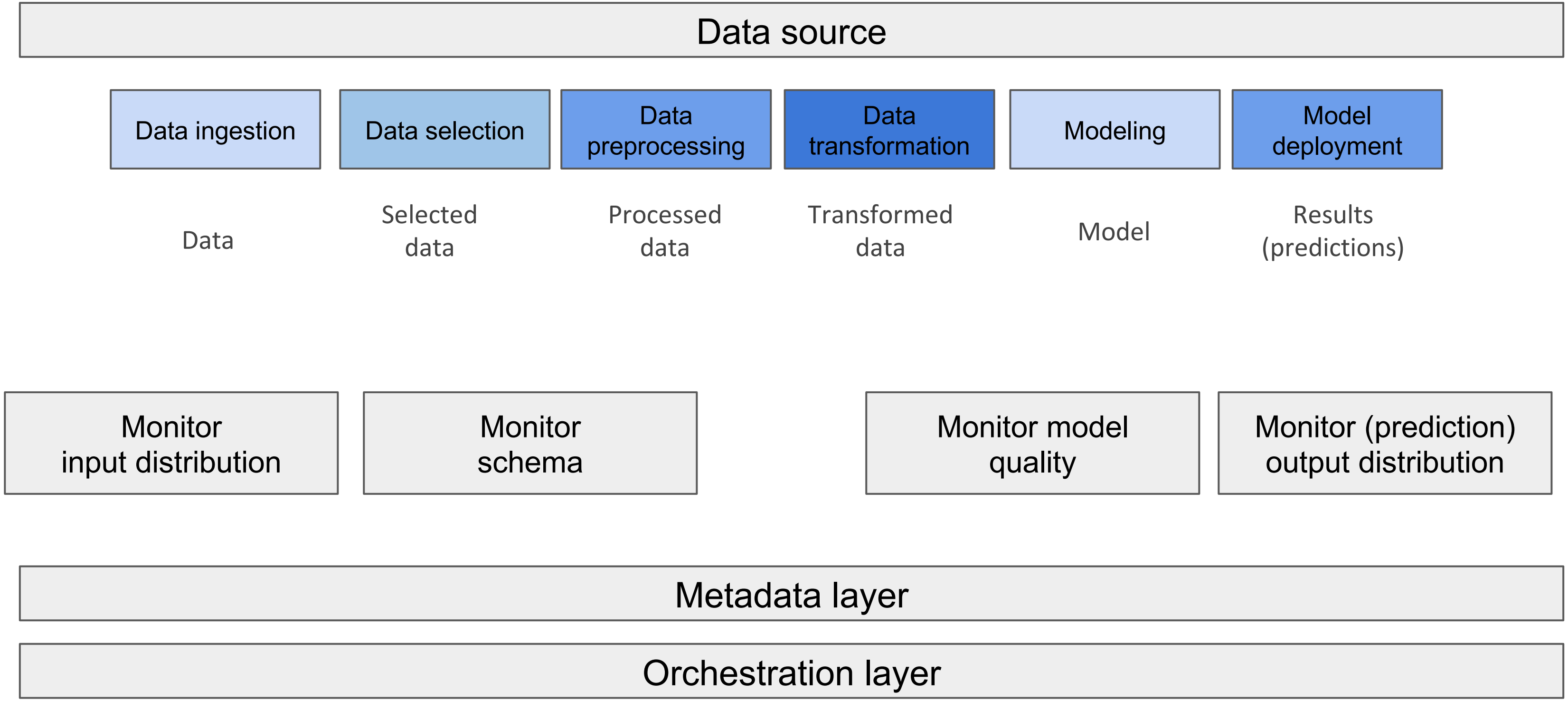
Retrieve and manage data from multiple sources

Data ETL (Extract Transform Load) pipelines

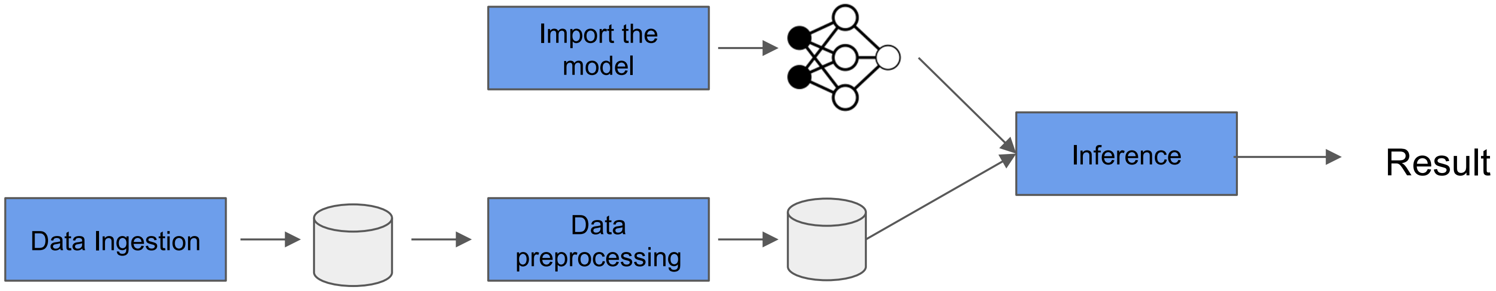
ML pipeline in production - Data



ML pipeline in production - Data



Orchestrator



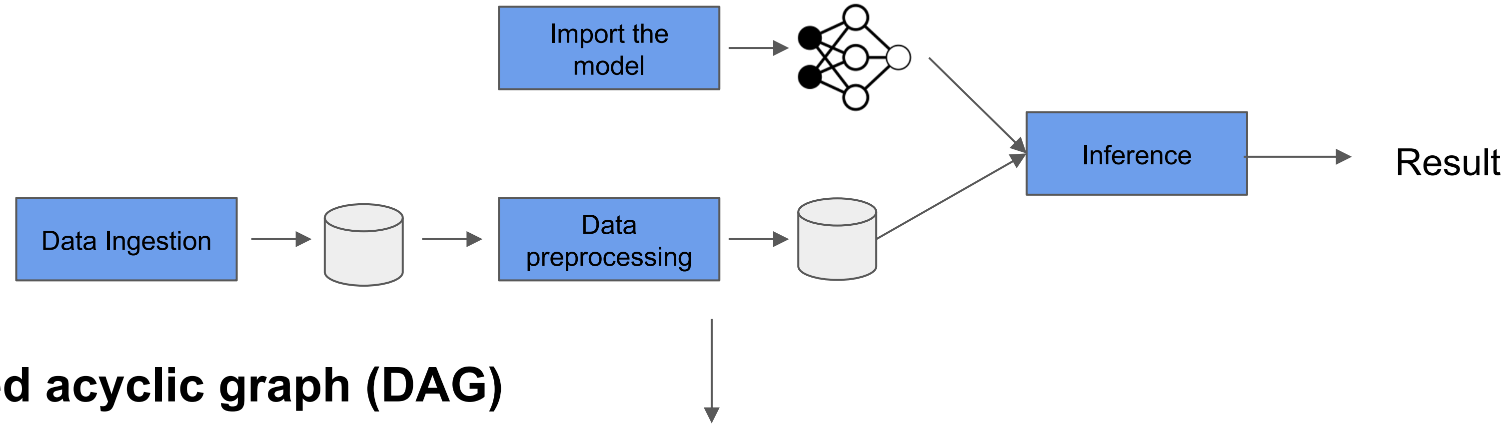
ML Pipeline → workflow

A **workflow** consists of an orchestrated and repeatable sequence of tasks, executed sequentially and/or concurrently

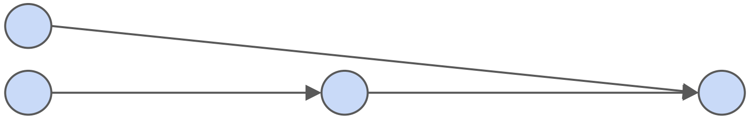
Orchestrator is a **workflow management system** that

- Build, connect, and maintain complex workflows

Orchestrator



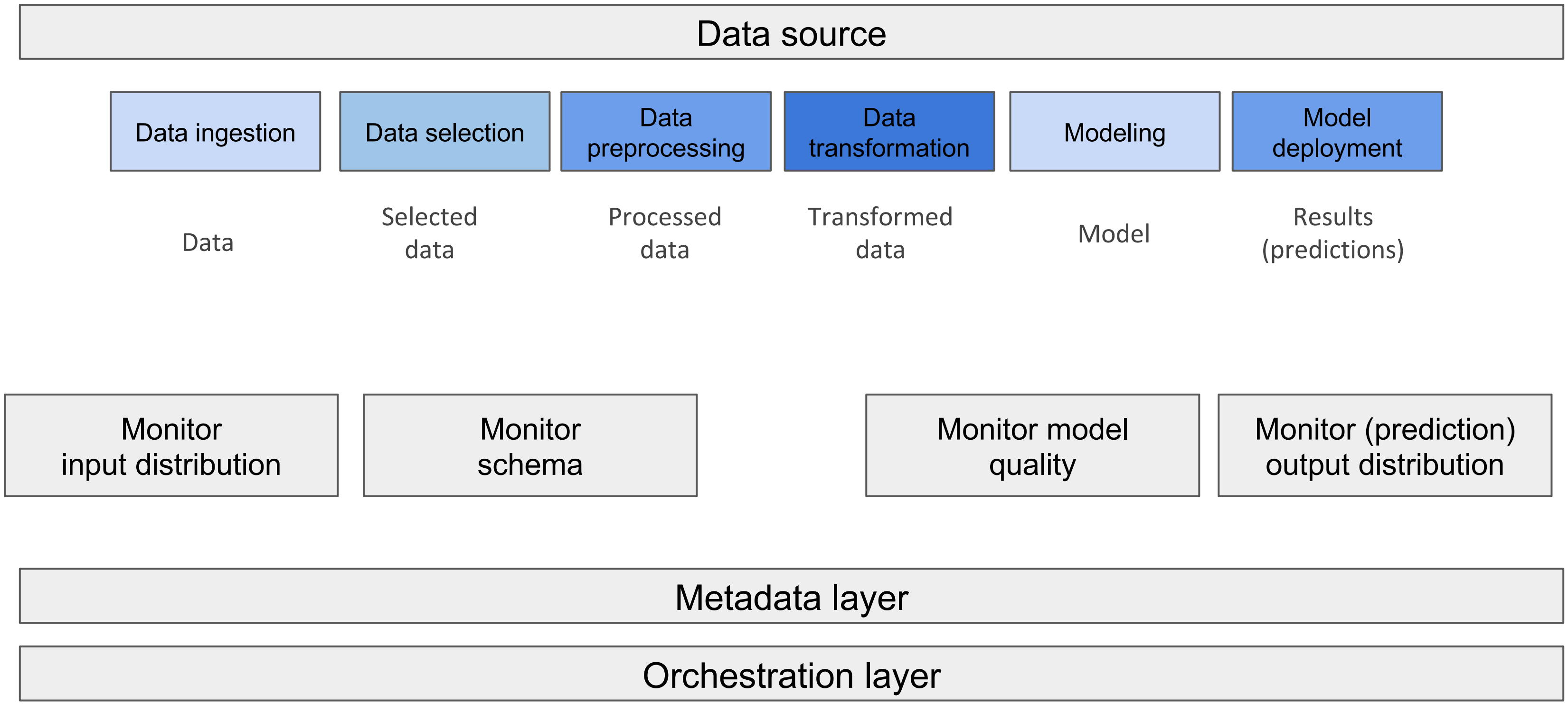
Directed acyclic graph (DAG)



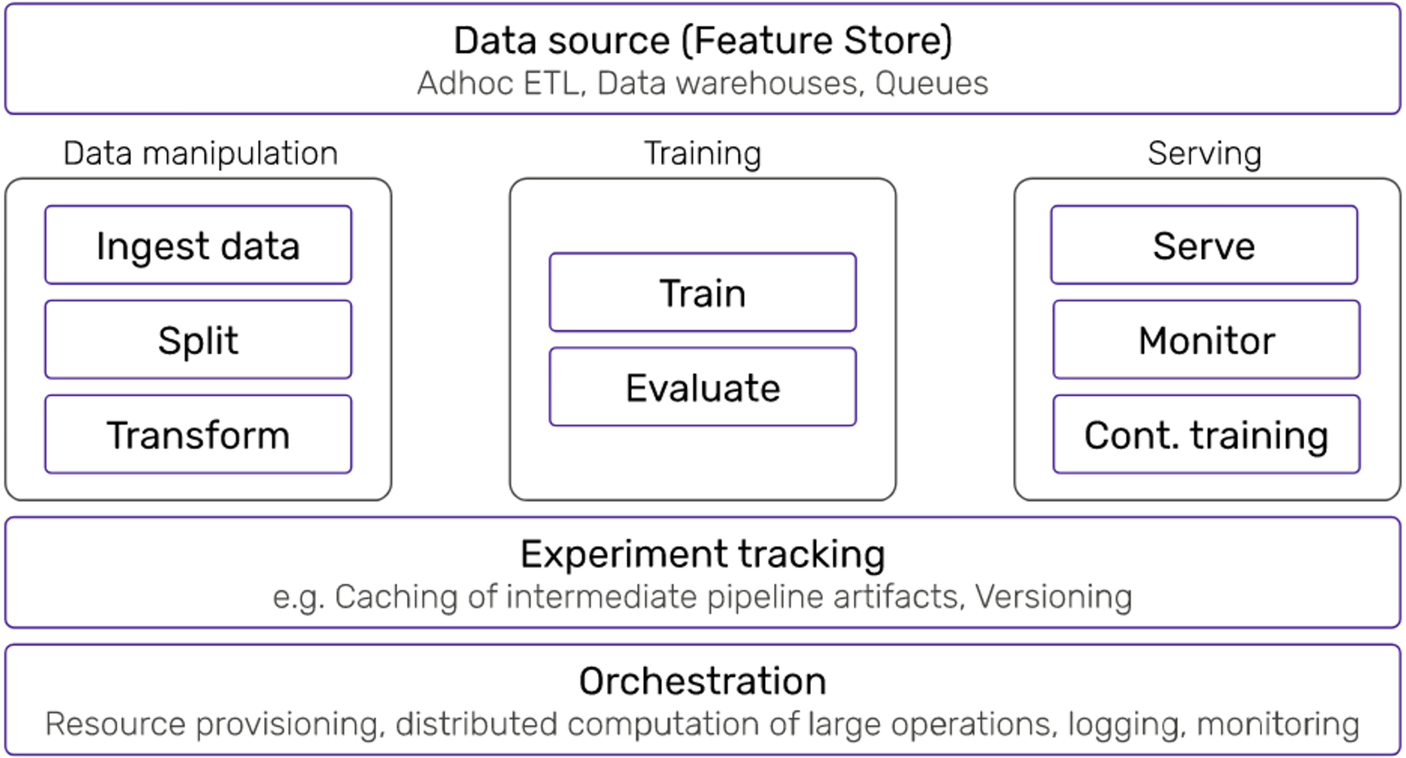
Steps of the pipeline form a DAG

- Model task relationships and dependencies

ML pipeline in production

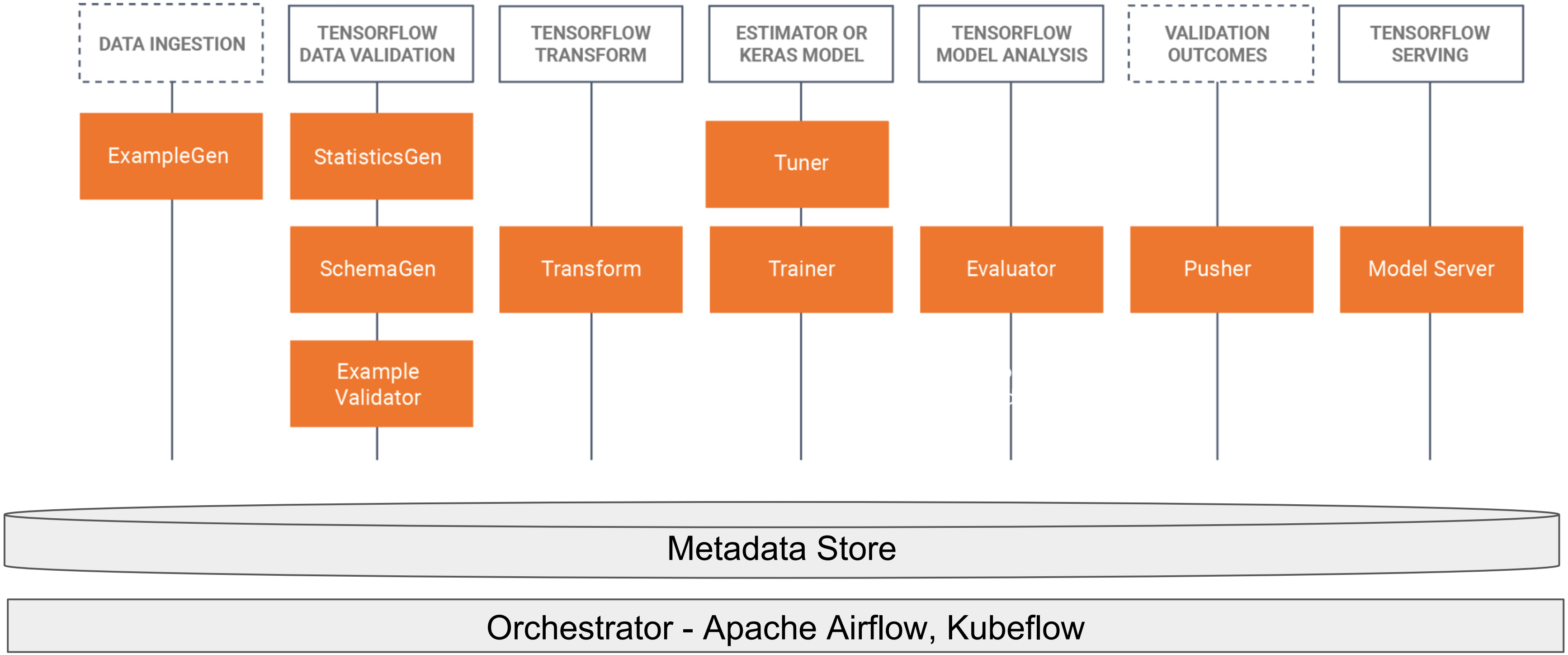


Production-ready ML architecture



https://blog.maiot.io/

TFX - Tensorflow Extended



https://www.tensorflow.org/tfx

Orchestration

Workflow management tools

- Apache Airflow (Airbnb)
- **Luigi** (Spotify)
- Kubeflow (Google)
- Azkaban (LinkedIn)

Luigi

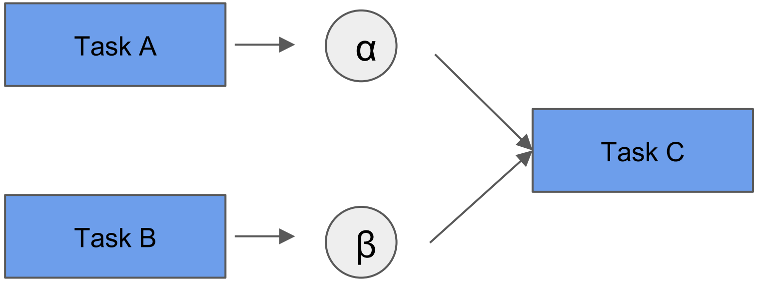
Open source workflow management system developed by Spotify

Luigi addresses all the “plumbing” associated with a data pipeline

- Chain many tasks
- Automate the task running
- Handle failures

The steps of a workflow are **task**, single unit of work

Luigi properties

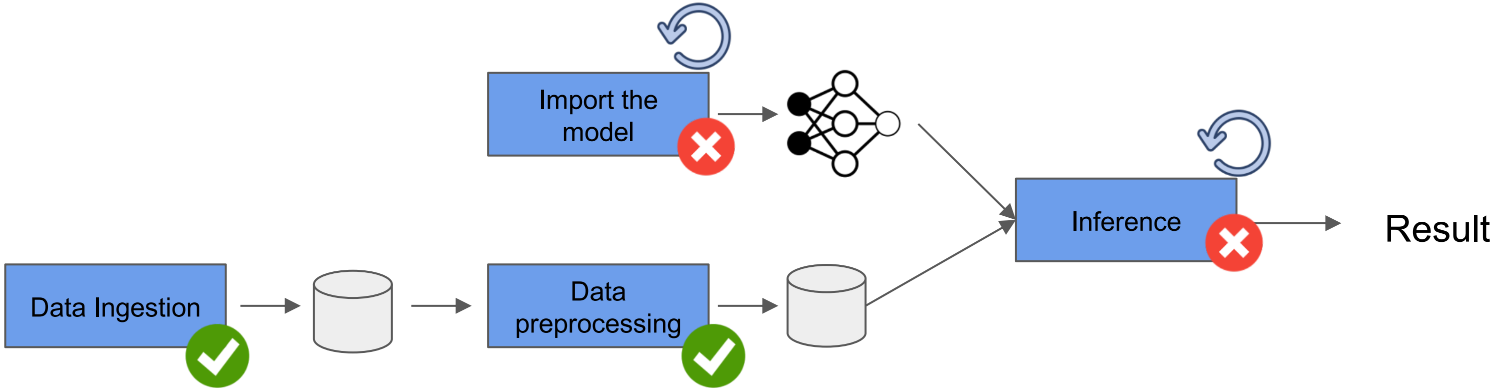


- Workflow management
- Dependency resolution
- Persistence of task state
- Idempotency property
  - Completed tasks are not run twice
- Re-use previously computed outputs
  - Automatically, without manually specifying it
- Failure management
  - Smoothly resume data workflow after a failure.

Failure management

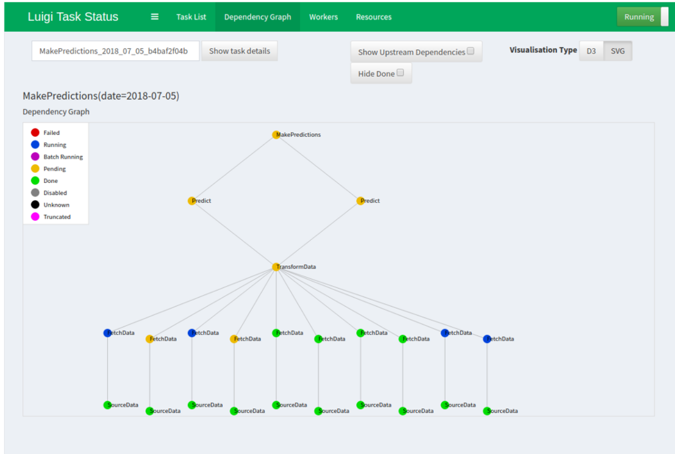
In case of failure of a component of the pipeline, the system is able to

- detect from which part of the workflow run again
- resume dependent task from the intermediate step



Luigi properties

- Lazy evaluation
  - delays the evaluation of task dependencies and workflow until its target is needed and avoids repeated evaluations
- Visualization
  - Graphical representation of the progress of the task in the data pipeline
- Parametrize and re-run tasks on a schedule with the help of an external trigger
- Command line integration
- Simple
  - Small overhead for a task
  - Connecting components is easy and intuitive.
- Python based



Luigi - Cons

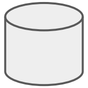
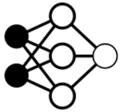
- No scheduler

- Luigi doesn't sync tasks to workers for you, schedule, alert, or monitor like other tools do (as Apache Airflow).

- It also has no native support for distributed execution.

Luigi building blocks

- Task 

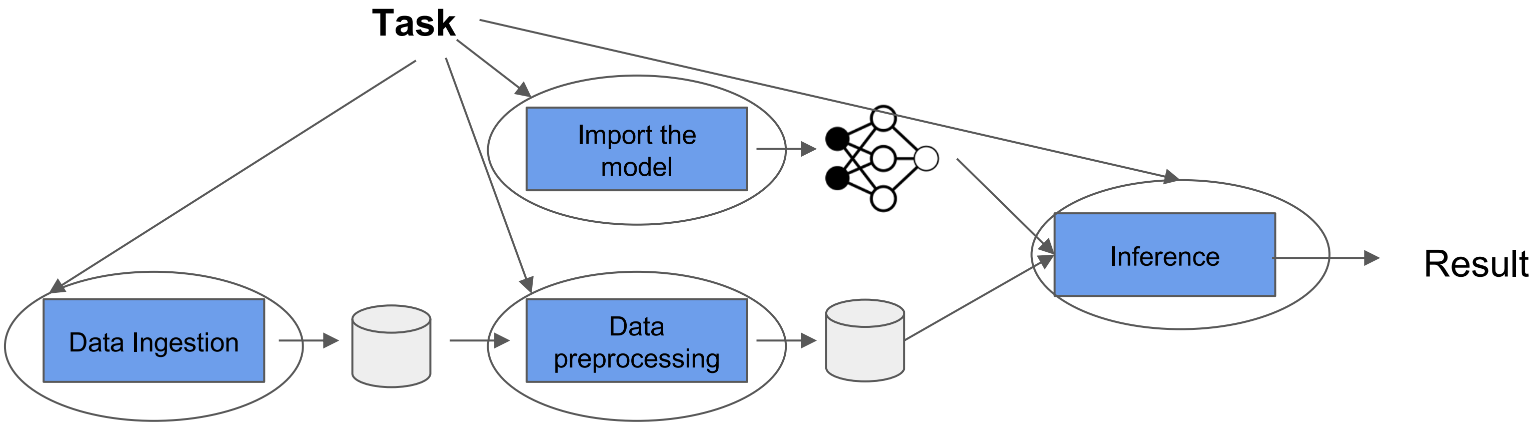
- Target  

- Parameter

Luigi building blocks

Task

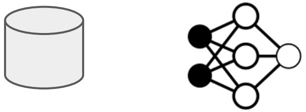
- Each step of a workflow
  - Usually a single unit of work
- Where computation is done
- Consume and produce targets
  - property of atomicity → recommended to output just one target



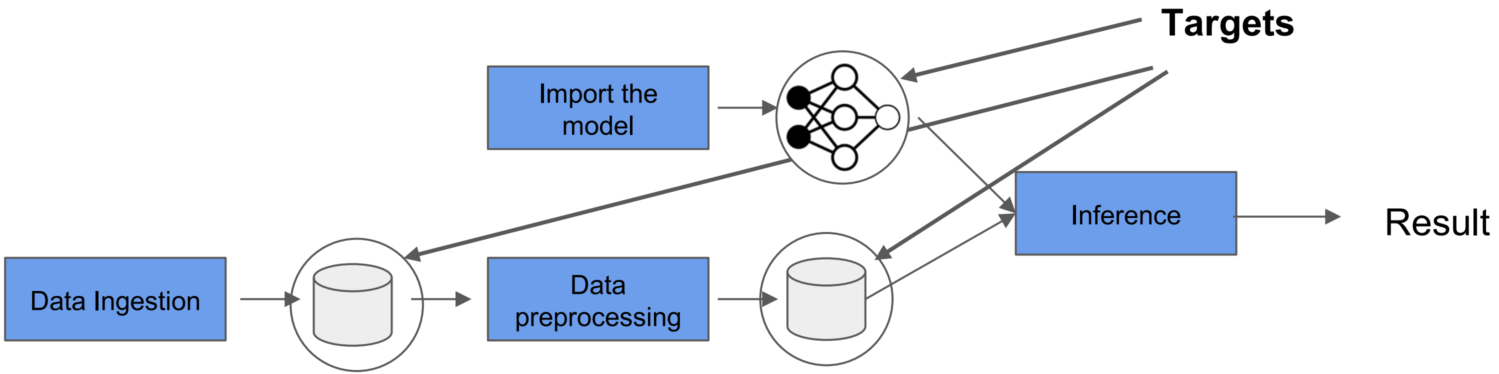
Luigi building blocks

Target

- Any kind of output generated by the task
    - e.g. a file, a checkpoint of the workflow



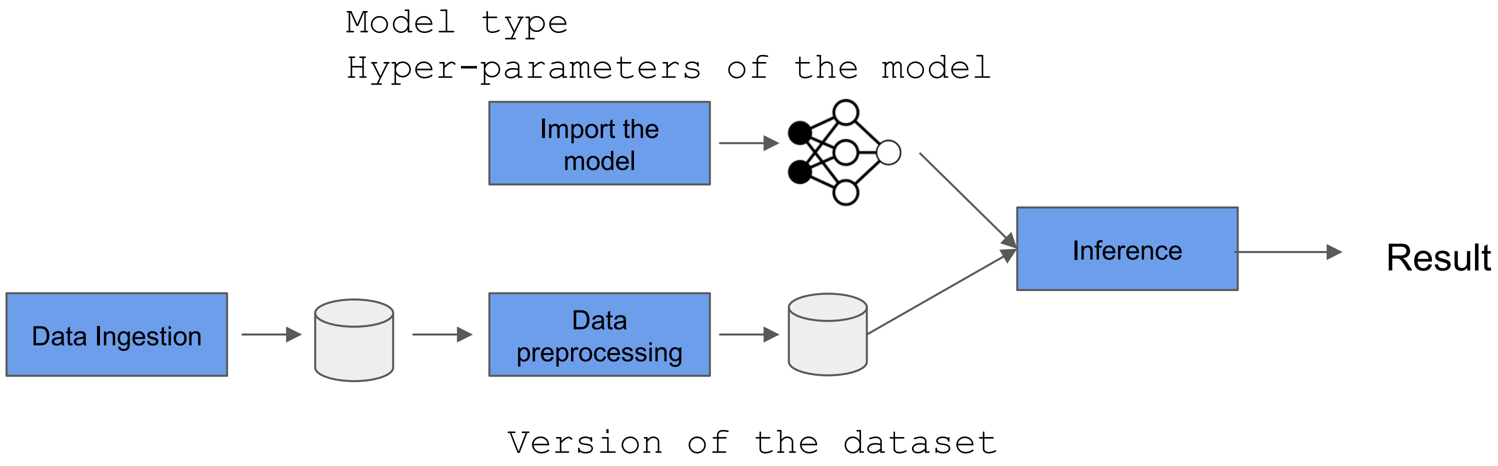
- Connect the task in the workflow



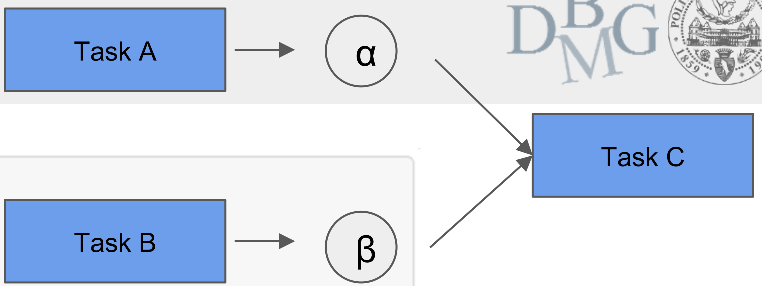
Luigi building blocks

Parameters

- Parameters of Task to perform parameterize tasks.
    - E.g. versions, hyper-parameters of module



Luigi - define Tasks




```python
import luigi

class MyTask(luigi.Task):
    param = luigi.Parameter(default=42)

    def requires(self):
        return SomeOtherTask(self.param)

    def run(self):
        f = self.output().open('w')
        print >>f, "hello, world"
        f.close()

    def output(self):
        return luigi.LocalTarget('/tmp/foo/bar-%s.txt' % self.param)

if __name__ == '__main__':
    luigi.run()
```

The business logic of the task

Where it writes output

What other tasks it depends on

Parameters for this task

https://luigi.readthedocs.io/en/stable/tasks.html

Run Luigi workflows

Use the command line specifying the module name and the task in the project directory

```
$ luigi --module <module_name> <task_name> --
<parameter1_name> <par_value> ..
```

To execute the entire workflow, we specify the last task

`<modul_name>` needs to be in our PYTHONPATH

We can add the current working directory to the PYTHONPATH with

```
PYTHONPATH='.' luigi --module <modul_name>...
```

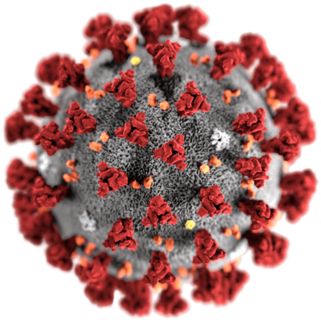
Luigi - Case Study

Case study on COVID-19 data

https://github.com/elianap/luigi-covid-pipeline

Objectives

- Create a daily report of COVID-19 situation in Italy
- Weekly model and predict for the next week the trend of variables of interest



Luigi - COVID-19 ML pipeline

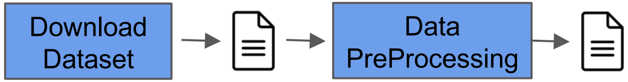
- Import data
- Pre-process data
- To generate a daily report
  - Generate some statistics or plots

    In our example → plot of the trend of some variable of interest
  - Aggregate in a single report

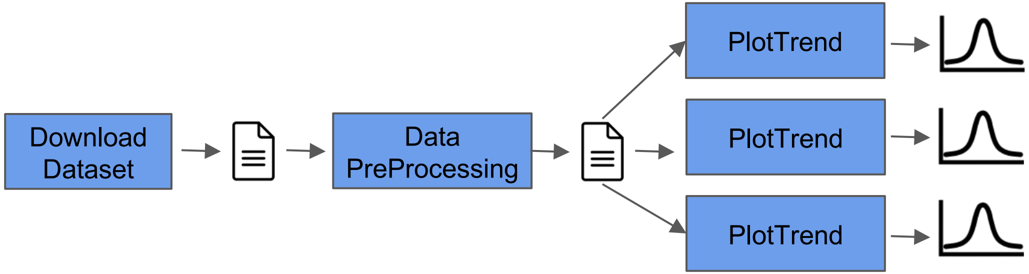
    In our example → single html page
- To weekly model and predict trends
  - Transform the data
  - Model (regression)
  - Predict the trend for a variable of interest
  - Plot the trend

Luigi - COVID-19 ML pipeline - DAG

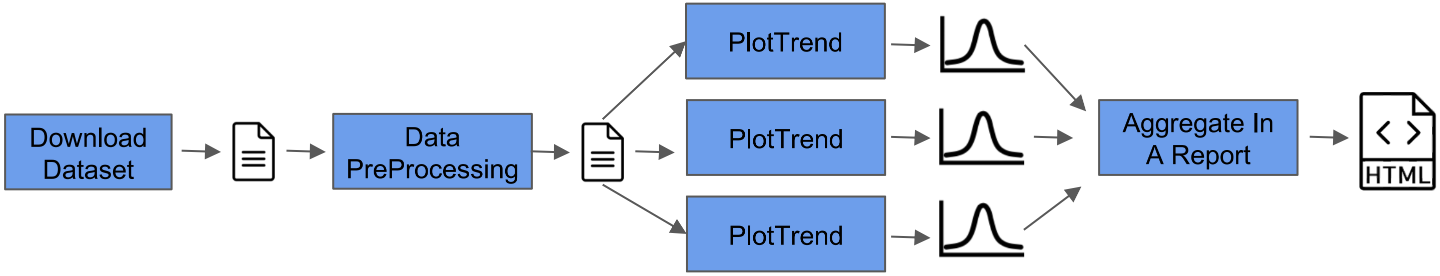
Luigi - COVID-19 ML pipeline - DAG



Luigi - COVID-19 ML pipeline - DAG

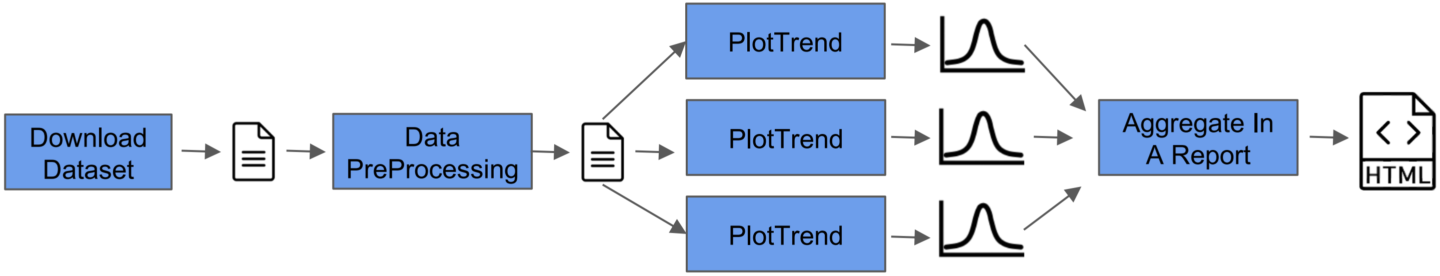


Luigi - COVID-19 ML pipeline - DAG

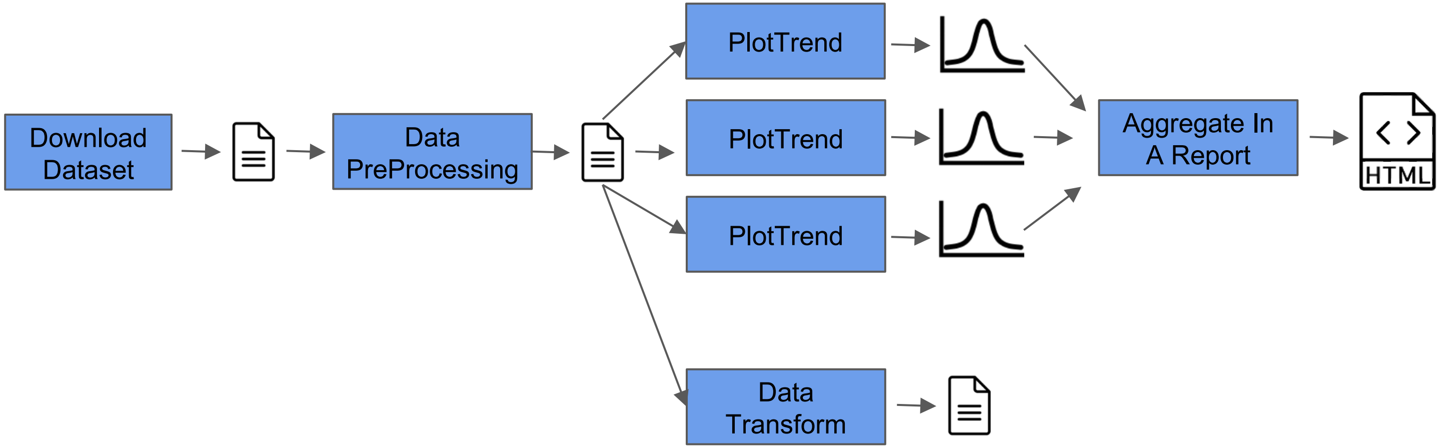


Luigi - COVID-19 ML pipeline - DAG

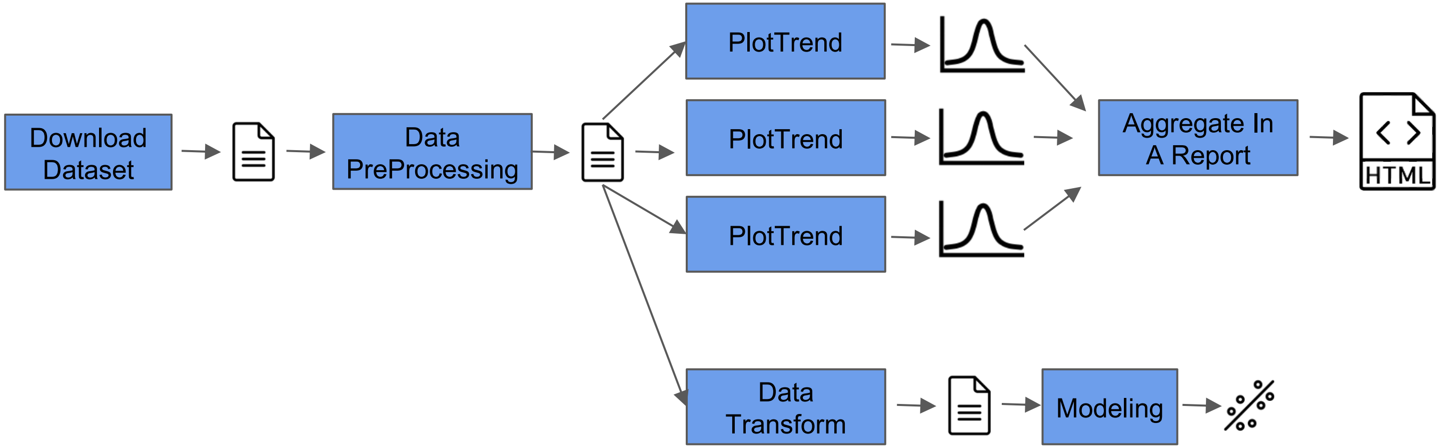
Daily report of the trend



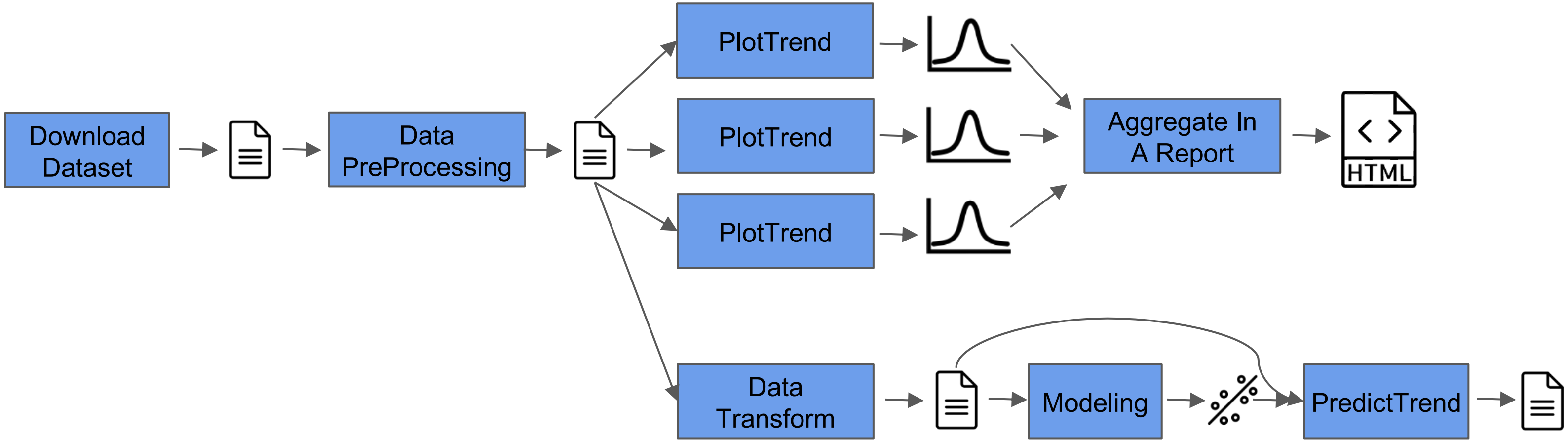
Luigi - COVID-19 ML pipeline - DAG



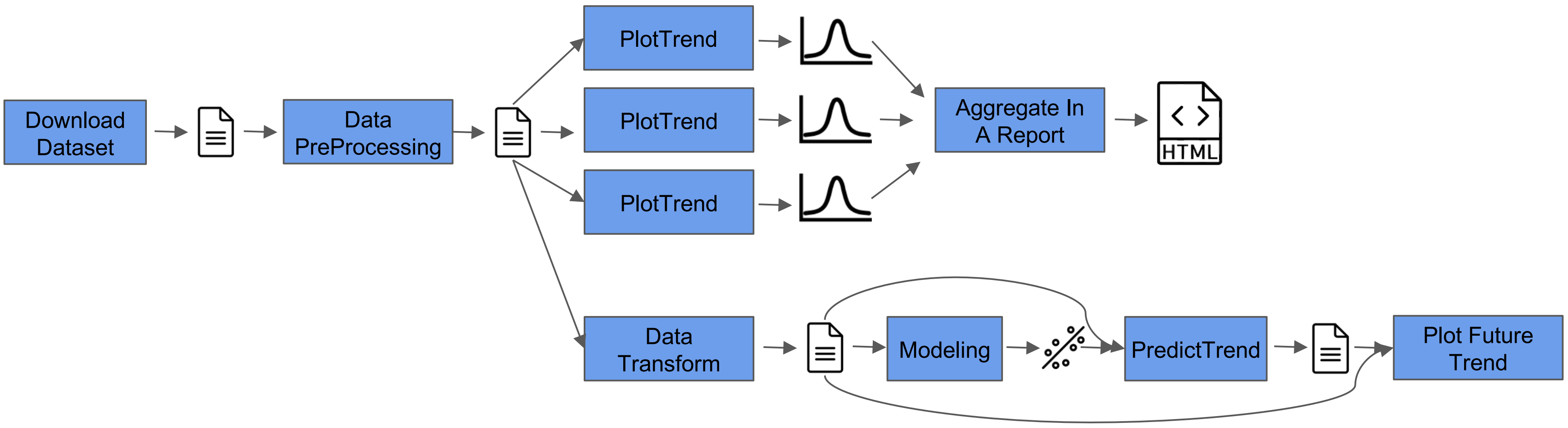
Luigi - COVID-19 ML pipeline - DAG



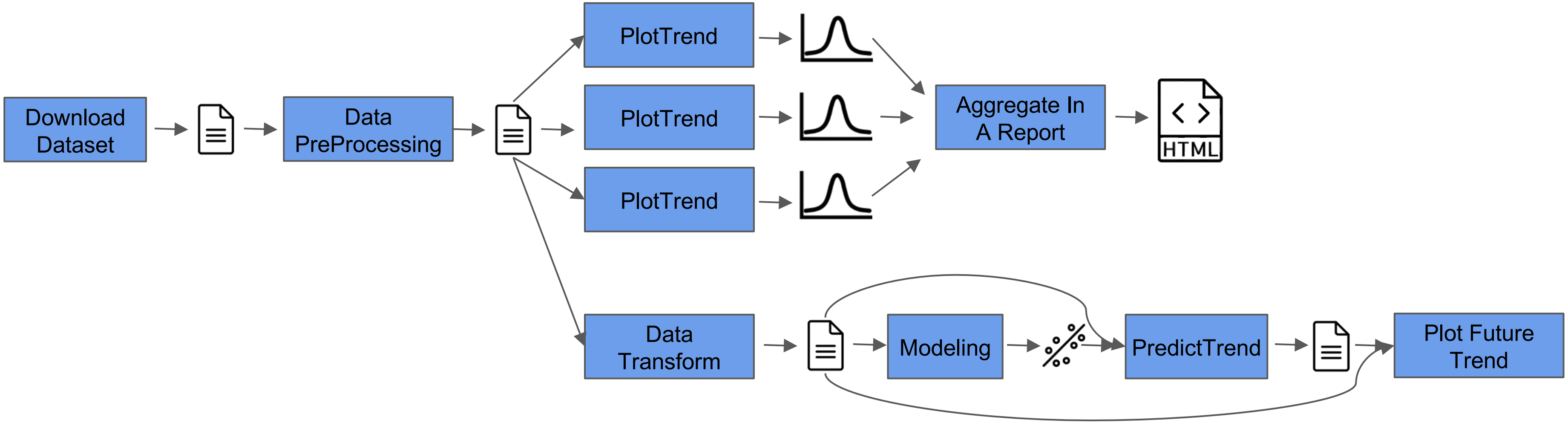
Luigi - COVID-19 ML pipeline - DAG



Luigi - COVID-19 ML pipeline - DAG



Luigi - COVID-19 ML pipeline - DAG



Weekly model and predict trends

Luigi - COVID-19 ML pipeline - DAG

DownloadDataset

```
class DownloadDataset(luigi.Task):

    dataset_version = DateParameter(default=datetime.date.today())
    dataset_name = Parameter(default="covidIT")


    columns_ita_eng = {"data": "date", "stato": "country",... }
    data_url =  "https://raw.githubusercontent.com/pcm-dpc/COVID-19/master/dati-
andamento-nazionale/dpc-covid19-ita-andamento-nazionale.csv"
    output_folder = os.path.join(output_dir, "dataset")
```

DownloadDataset

```python
class DownloadDataset(luigi.Task):

    dataset_version = DateParameter(default=datetime.date.today())
    dataset_name = Parameter(default="dataset")
    ....

    def output(self):
        return LocalTarget( os.path.join(self.output_folder,
                           f"{self.dataset_name}_v{self.dataset_version}.csv"
        ))
```

DownloadDataset

```
class DownloadDataset(luigi.Task):

    dataset_version = DateParameter(default=datetime.date.today())
    dataset_name = Parameter(default="dataset")
    ....

    def run(self):
        df_data = self.load_data(self.data_url, columns_new_names=self.columns_eng)

        Path(self.output_folder).mkdir(parents=True, exist_ok=True)

        df_data.to_csv(self.output().path)
```

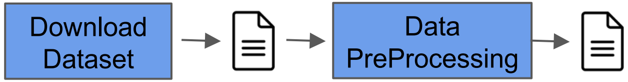
DownloadDataset

```python
class DownloadDataset(luigi.Task):

    dataset_version = DateParameter(default=datetime.date.today())
    dataset_name = Parameter(default="dataset")
    ....

    def load_data(self, data_url, columns_new_names=None):
        data = pd.read_csv(data_url)
        if columns_new_names:
            data.rename(columns=columns_new_names, inplace=True)
        data["date"] = pd.to_datetime(data["date"])
        data.set_index("date", inplace=True)
        return data
```

Run DownloadDataset Luigi task

```
PYTHONPATH='.' luigi --module covid_pipeline DownloadDataset
```

Luigi - COVID-19 ML pipeline - DAG



DataPreProcessing

```
class DataPreProcessing(luigi.Task):

    dataset_version = DateParameter(default=datetime.date.today())
    dataset_name = Parameter(default="dataset")
    output_folder = os.path.join(output_dir, "processed")

    def requires(self):
        return DownloadDataset(self.dataset_version, self.dataset_name)
```

DataPreProcessing

```python
class DataPreProcessing(luigi.Task):
…

    def output(self):
        return LocalTarget( os.path.join( self.output_folder,
            f"{self.dataset_name}_processed_v{self.dataset_version}.csv"
        ))
```

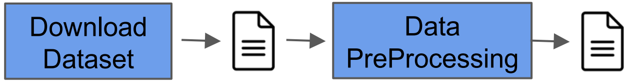
DataPreProcessing

```
class DataPreProcessing(luigi.Task):
…
    def run(self):
        df_data = pd.read_csv(self.input().path, index_col="date")
        df_data = self.preprocess_data(df_data)

        Path(self.output_folder).mkdir(parents=True, exist_ok=True)

        df_data.to_csv(self.output().path)
```

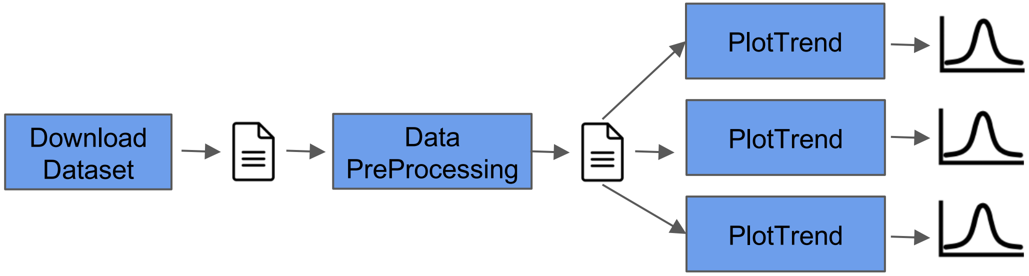
DataPreProcessing

```
class DataPreProcessing(luigi.Task):
…

    def preprocess_data(self, df_data):
        df_data["diff_death"] = df_data["death"].diff()
        df_data["diff_intensive_care"] = df_data["intensive_care"].diff()
        df_data["diff_performed_tests"] = df_data["performed_tests"].diff()
        df_data["diff_recovered"] = df_data["recovered"].diff()
        df_data["ratio_molecular"] = ( df_data["total_positives_molecular_test"] /
                                       df_data["swabs_test_molecular"]
        df_data["ratio_antigenic"] =df_data["total_positives_antigenic_test_rapid"]/
                                    df_data["swabs_test_antigenic_rapid"]
        return df_data
```

Luigi - COVID-19 ML pipeline - DAG

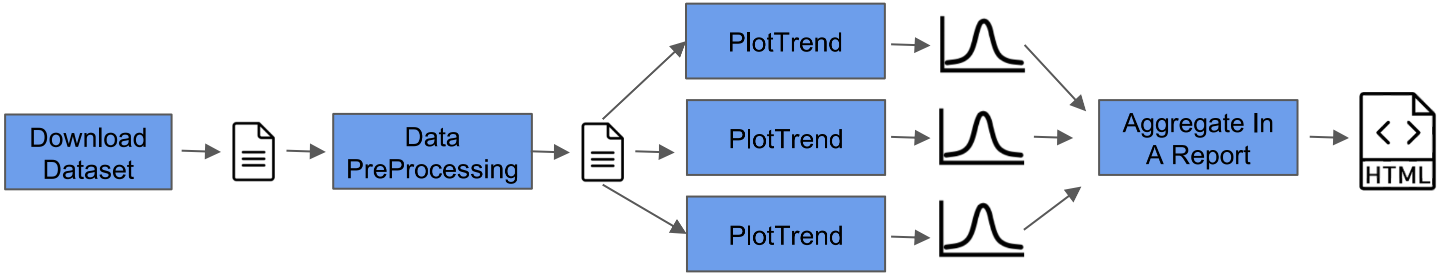


Luigi - COVID-19 ML pipeline - DAG



Luigi - COVID-19 ML pipeline - DAG

Daily report of the trend



AggregateInReport

```
class AggregateInReport(luigi.Task):
...
    # --> Alternative for dynamic report -->
     run as --attributes '[ "total_positive", "recovered", "ratio_molecular" ]'
    # attributes = ListParameter(default=["total_positive",
     "recovered", "ratio_molecular"])
    #

    attributes = ["total_positive", "recovered","ratio_molecular"]


    def output(self):
            return LocalTarget( os.path.join( self.output_folder,
     f"{self.dataset_name}_report_trends_v{self.dataset_version}.html"))
```

AggregateInReport

```
class AggregateInReport(luigi.Task):
...
    def requires(self):
        return {
            attribute: PlotTrend(self.dataset_version, self.dataset_name, attribute)
            for attribute in self.attributes
        }

    def run(self):
        path_by_attribute = {k: self.input()[k].path for k in self.input()}
        plots_html = self.getHTMLTrends(path_by_attribute)
        ... #write in file
        with open(self.output().path, "w") as fp:
            for plot_html in plots_html:
                fp.write(plot_html)
```

AggregateInReport

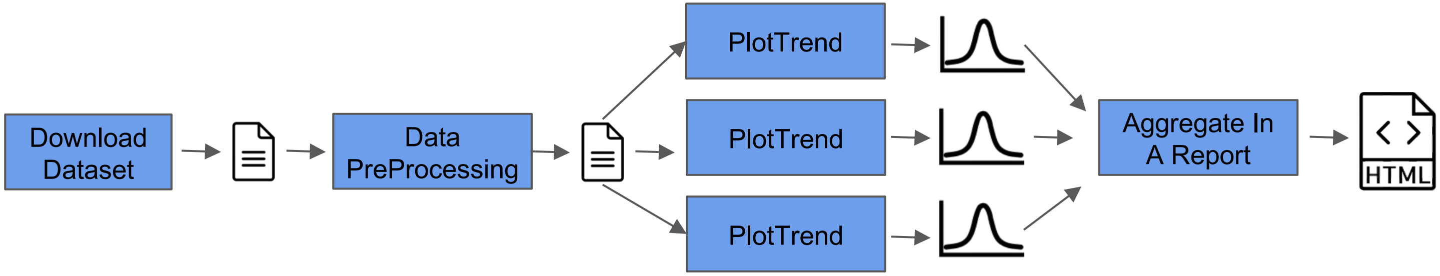
```python
class AggregateInReport(luigi.Task):
...
    def requires(self):
        return {
            attribute: PlotTrend(self.dataset_version, self.dataset_name, attribute)
            for attribute in self.attributes
        }

    def run(self):
        path_by_attribute = {k: self.input()[k].path for k in self.input()}

        plots_html = self.getHTMLTrends(path_by_attribute)
        ... #write in file
        with open(self.output().path, "w") as fp:
            for plot_html in plots_html:
                fp.write(plot_html)
```

AggregateInReport

```python
class AggregateInReport(luigi.Task):
...
    def requires(self):
        return {
            attribute: PlotTrend(self.dataset_version, self.dataset_name, attribute)
            for attribute in self.attributes
        }

    def getHTMLTrends(self, path_by_attribute):
        plots_html = [
            f"<h2 style='text-align: center'>{k}</h2>\n<p style='text-align:
center'><img src='{path_by_attribute[k]}'  style='width: 50%; height: 50%' /> </p>"
            for k in path_by_attribute
        ]
        return plots_html
```

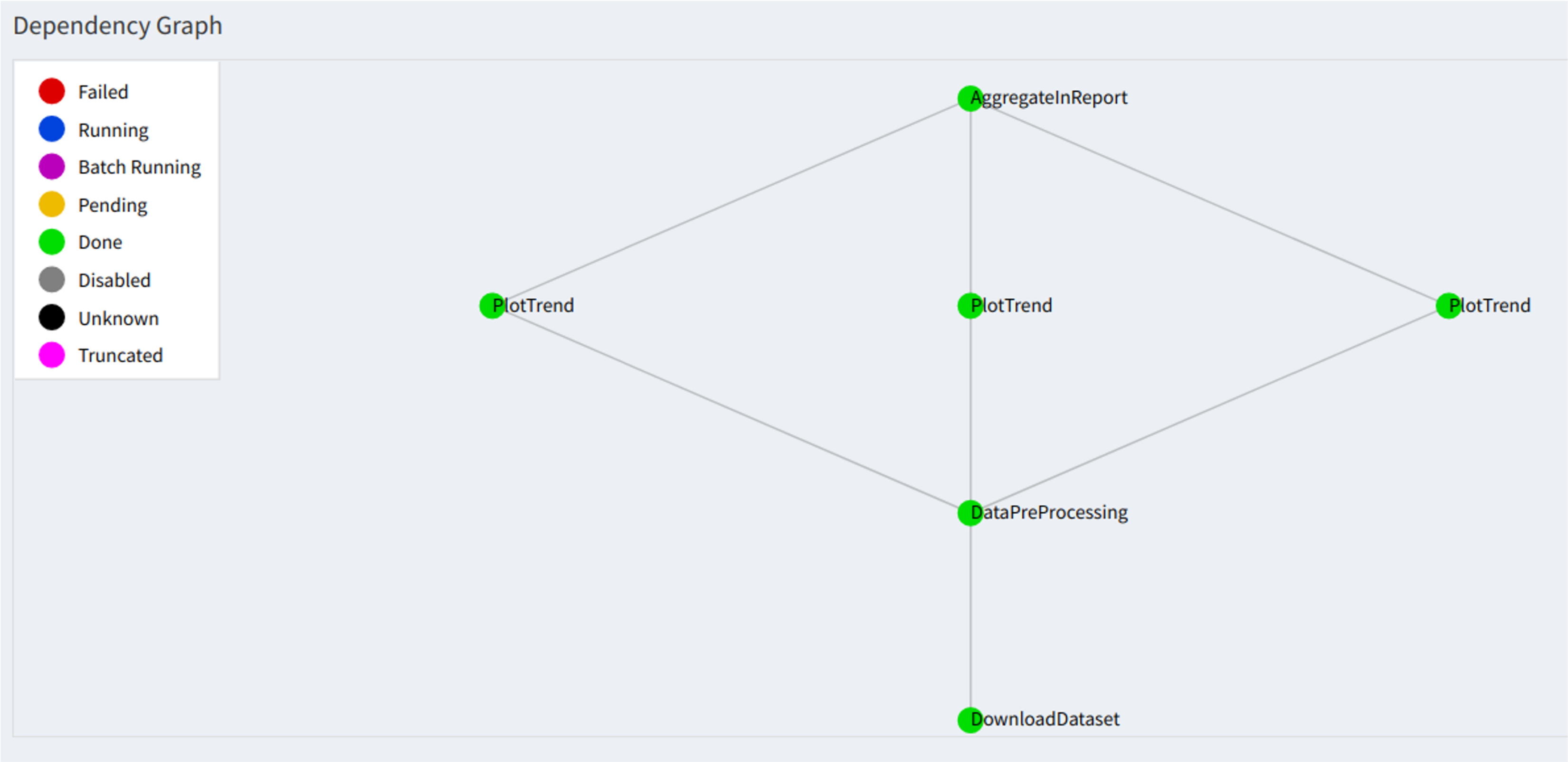
Luigi - COVID-19 ML pipeline - DAG

Daily report of the trend

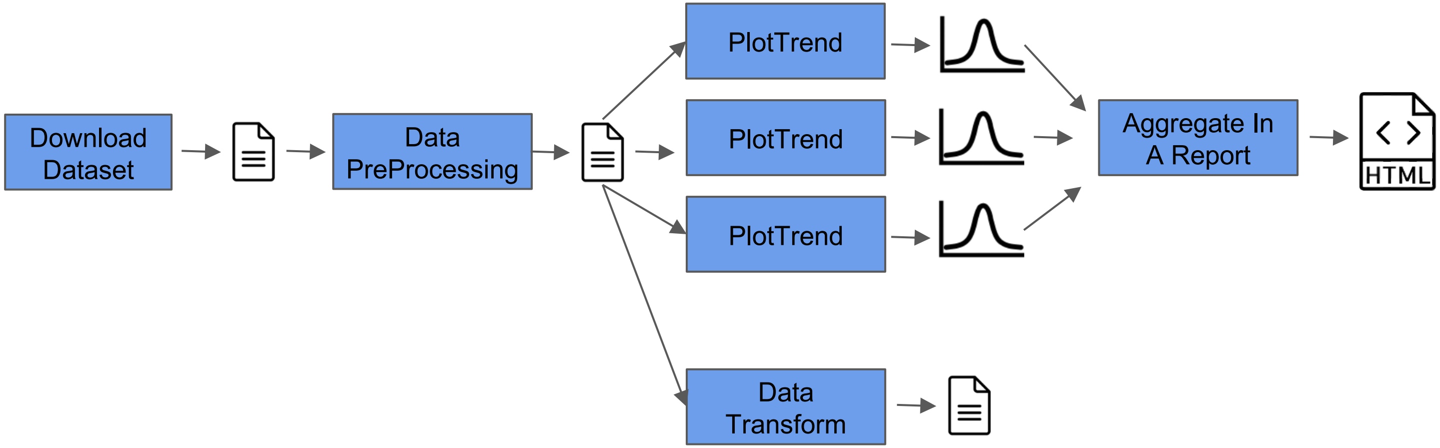


AggregateInReport

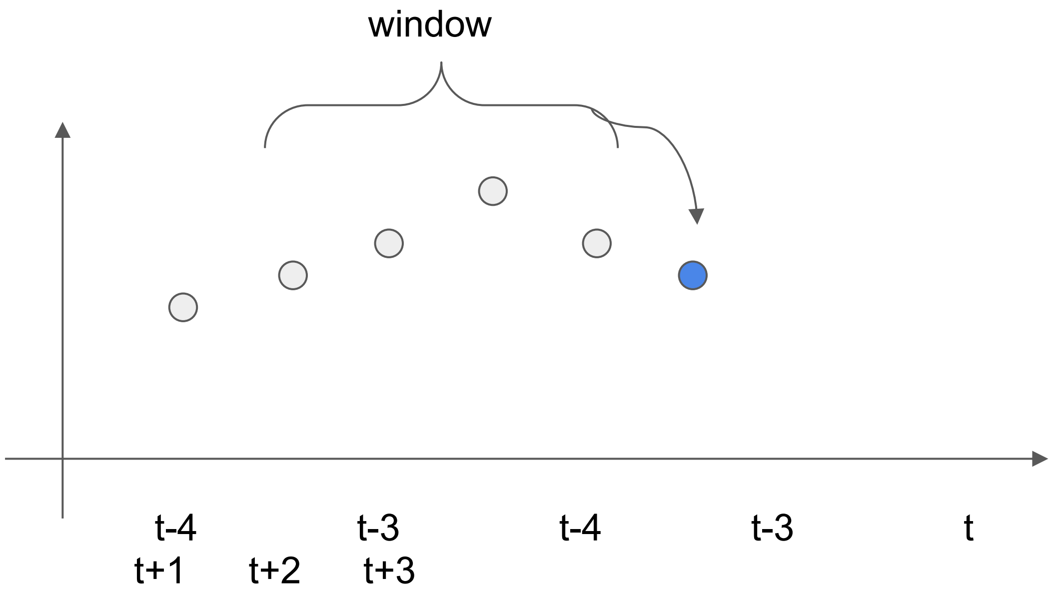
Luigi Task Visualizer http://localhost:8082



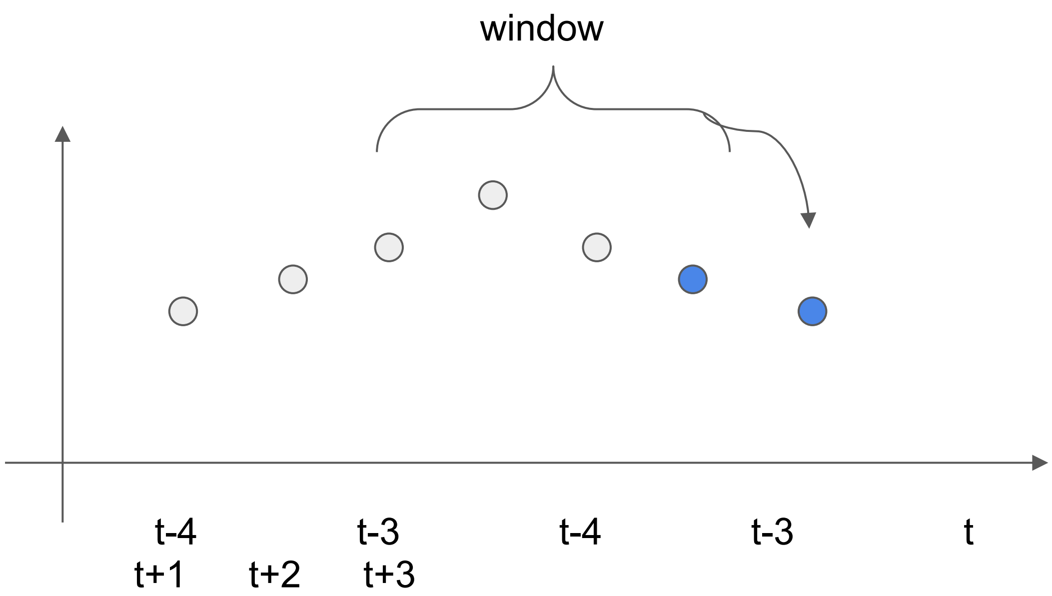
Luigi - COVID-19 ML pipeline - DAG



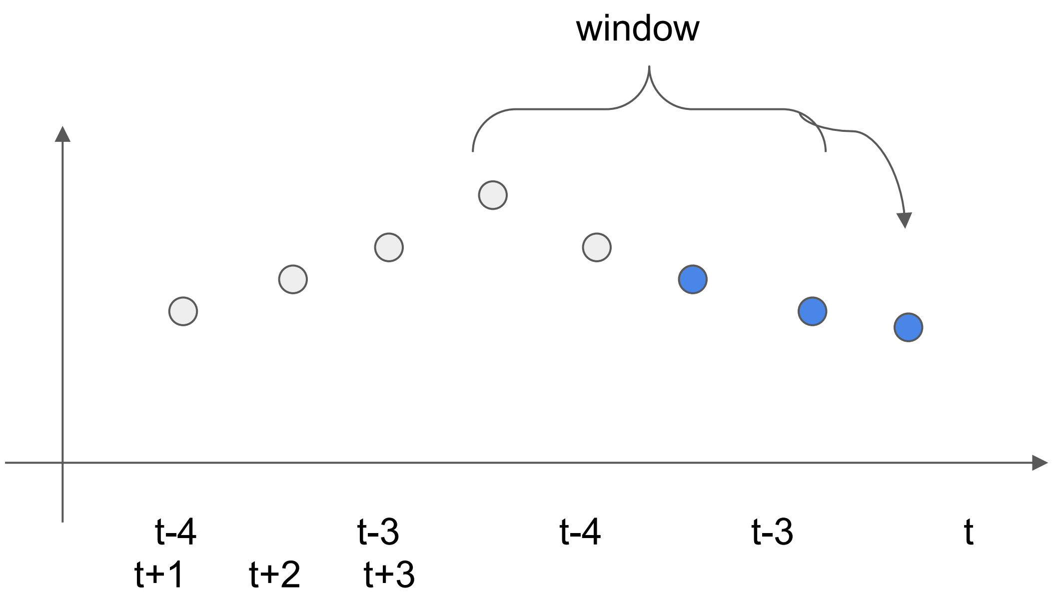
DataTransform - Windowing



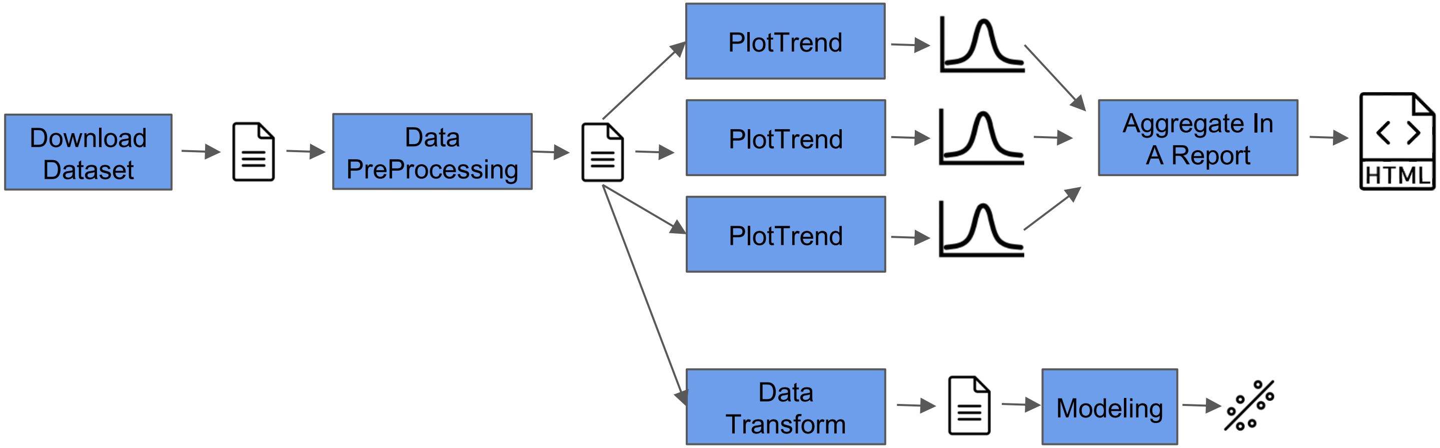
DataTransform - Windowing



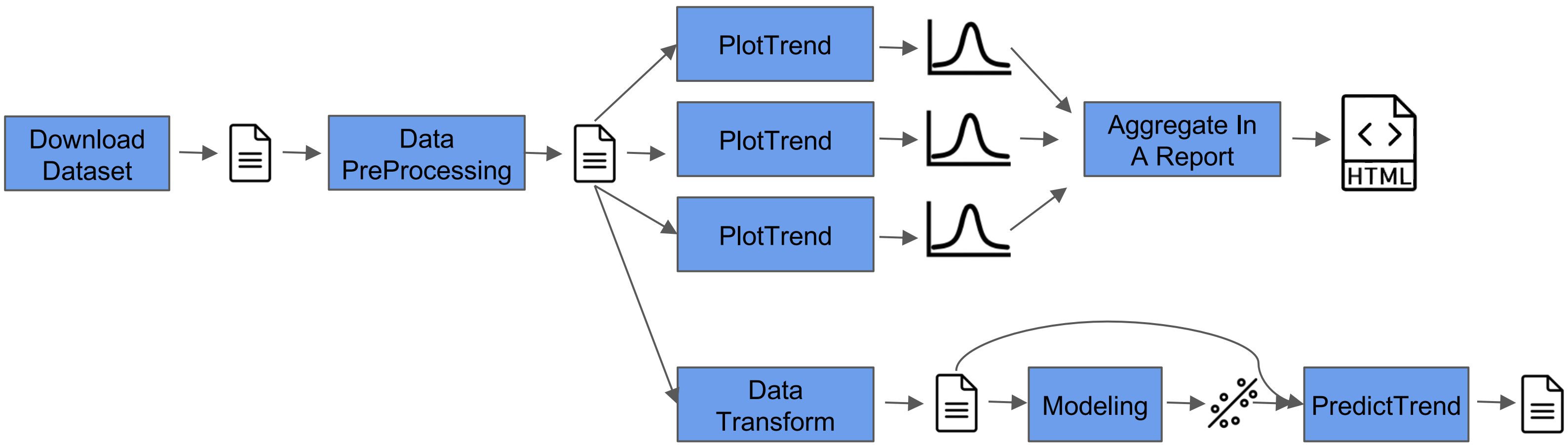
DataTransform - Windowing



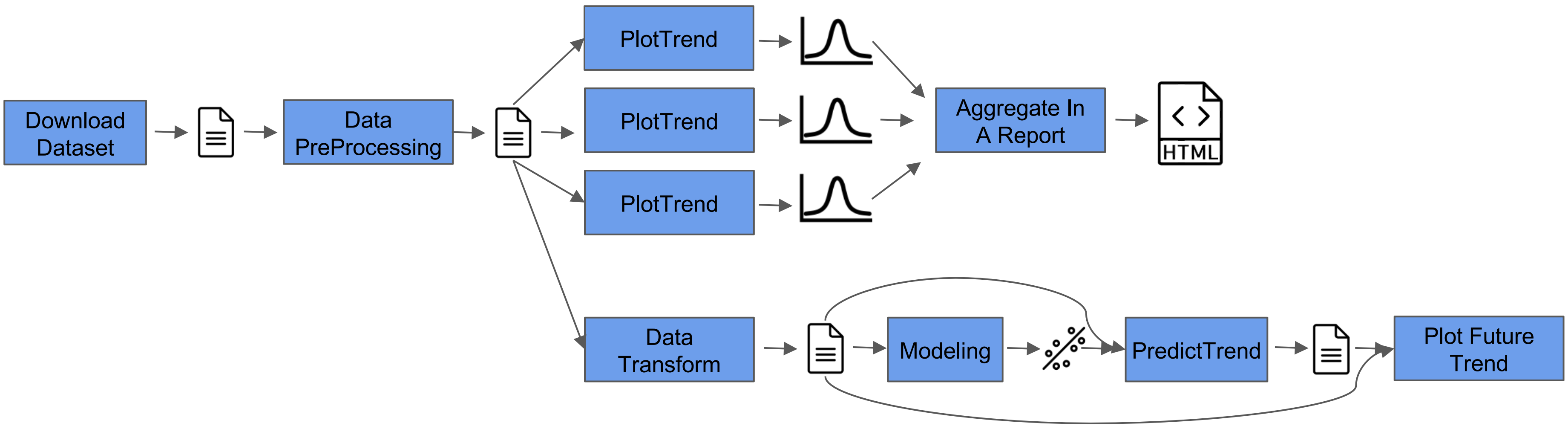
Luigi - COVID-19 ML pipeline - DAG



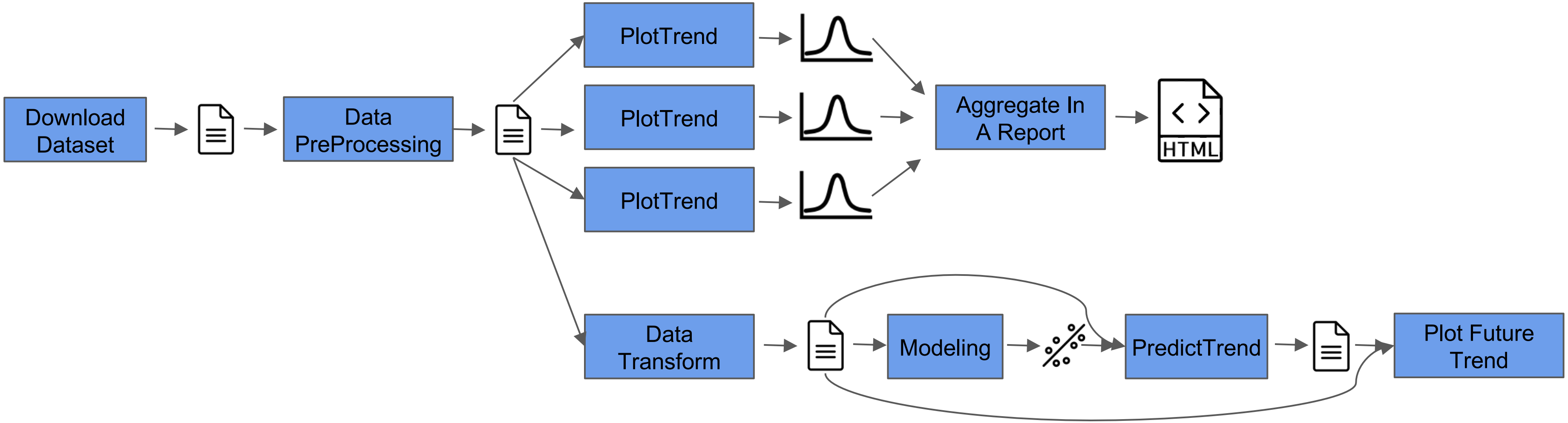
Luigi - COVID-19 ML pipeline - DAG



Luigi - COVID-19 ML pipeline - DAG



Luigi - COVID-19 ML pipeline - DAG



Weekly model and predict trends