C2_W3_Lab_2_IterativeSchema

August 20, 2022

1 Ungraded Lab: Iterative Schema with TFX and ML Metadata

In this notebook, you will get to review how to update an inferred schema and save the result to the metadata store used by TFX. As mentioned before, the TFX components get information from this database before running executions. Thus, if you will be curating a schema, you will need to save this as an artifact in the metadata store. You will get to see how that is done in the following exercise.

Afterwards, you will also practice accessing the TFX metadata store and see how you can track the lineage of an artifact.

1.1 Setup

1.1.1 Imports

[1]:
```
import tensorflow as tf
import tensorflow_data_validation as tfdv

from tfx import v1 as tfx

from tfx.types import standard_artifacts

from tfx.orchestration.experimental.interactive.interactive_context import␣
↪InteractiveContext
from google.protobuf.json_format import MessageToDict
from tensorflow_metadata.proto.v0 import schema_pb2

import os
import pprint
pp = pprint.PrettyPrinter()
```

1.1.2 Define paths

For familiarity, you will again be using the Census Income dataset from the previous weeks' ungraded labs. You will use the same paths to your raw data and pipeline files as shown below.

[2]:
```
# location of the pipeline metadata store
_pipeline_root = './pipeline/'
```

```
# directory of the raw data files
_data_root = './data/census_data'

# path to the raw training data
_data_filepath = os.path.join(_data_root, 'adult.data')
```

1.2 Data Pipeline

Each TFX component you use accepts and generates artifacts which are instances of the different artifact types TFX has configured in the metadata store. The properties of these instances are shown neatly in a table in the outputs of `context.run()`. TFX does all of these for you so you only need to inspect the output of each component to know which property of the artifact you can pass on to the next component (e.g. the `outputs['examples']` of `ExampleGen` can be passed to `StatisticsGen`).

Since you've already used this dataset before, we will just quickly go over `ExampleGen`, `StatisticsGen`, and `SchemaGen`. The new concepts will be discussed after the said components.

1.2.1 Create the Interactive Context

[3]:
```
# Initialize the InteractiveContext.
# If you leave `_pipeline_root` blank, then the db will be created in a␣
 ↪temporary directory.
context = InteractiveContext(pipeline_root=_pipeline_root)
```

```
WARNING:absl:InteractiveContext metadata_connection_config not provided: using
SQLite ML Metadata database at ./pipeline/metadata.sqlite.
```

1.2.2 ExampleGen

[4]:
```
# Instantiate ExampleGen with the input CSV dataset
example_gen = tfx.components.CsvExampleGen(input_base=_data_root)

# Execute the component
context.run(example_gen)
```

```
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.8
interpreter.
```

[4]:
```
ExecutionResult(
    component_id: CsvExampleGen
    execution_id: 1
    outputs:
        examples: Channel(
            type_name: Examples
            artifacts: [Artifact(artifact: id: 1
        type_id: 14
        uri: "./pipeline/CsvExampleGen/examples/1"
```

```
        properties {
          key: "split_names"
          value {
            string_value: "[\"train\", \"eval\"]"
          }
        }
        custom_properties {
          key: "file_format"
          value {
            string_value: "tfrecords_gzip"
          }
        }
        custom_properties {
          key: "input_fingerprint"
          value {
            string_value: "split:single_split,num_files:1,total_bytes:3974460,xo
r_checksum:1635234403,sum_checksum:1635234403"
          }
        }
        custom_properties {
          key: "payload_format"
          value {
            string_value: "FORMAT_TF_EXAMPLE"
          }
        }
        custom_properties {
          key: "span"
          value {
            int_value: 0
          }
        }
        custom_properties {
          key: "state"
          value {
            string_value: "published"
          }
        }
        custom_properties {
          key: "tfx_version"
          value {
            string_value: "1.3.0"
          }
        }
        state: LIVE
        , artifact_type: id: 14
        name: "Examples"
        properties {
```

```
    key: "span"
    value: INT
  }
  properties {
    key: "split_names"
    value: STRING
  }
  properties {
    key: "version"
    value: INT
  }
  )]
      additional_properties: {}
      additional_custom_properties: {}
  ))
```

1.2.3 StatisticsGen

[5]:
```
# Instantiate StatisticsGen with the ExampleGen ingested dataset
statistics_gen = tfx.components.StatisticsGen(
    examples=example_gen.outputs['examples'])

# Execute the component
context.run(statistics_gen)
```

```
WARNING:root:Make sure that locally built Python SDK docker image has Python 3.8
interpreter.
```

[5]:
```
ExecutionResult(
    component_id: StatisticsGen
    execution_id: 2
    outputs:
        statistics: Channel(
            type_name: ExampleStatistics
            artifacts: [Artifact(artifact: id: 2
        type_id: 16
        uri: "./pipeline/StatisticsGen/statistics/2"
        properties {
          key: "split_names"
          value {
            string_value: "[\"train\", \"eval\"]"
          }
        }
        custom_properties {
          key: "name"
          value {
            string_value: "statistics"
```

```
    }
  }
  custom_properties {
    key: "producer_component"
    value {
      string_value: "StatisticsGen"
    }
  }
  custom_properties {
    key: "state"
    value {
      string_value: "published"
    }
  }
  custom_properties {
    key: "tfx_version"
    value {
      string_value: "1.3.0"
    }
  }
  state: LIVE
  , artifact_type: id: 16
  name: "ExampleStatistics"
  properties {
    key: "span"
    value: INT
  }
  properties {
    key: "split_names"
    value: STRING
  }
  )]
      additional_properties: {}
      additional_custom_properties: {}
  ))
```

1.2.4 SchemaGen

[6]:
```
# Instantiate SchemaGen with the StatisticsGen ingested dataset
schema_gen = tfx.components.SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    )

# Run the component
context.run(schema_gen)
```

[6]:
```
ExecutionResult(
    component_id: SchemaGen
    execution_id: 3
    outputs:
        schema: Channel(
            type_name: Schema
            artifacts: [Artifact(artifact: id: 3
        type_id: 18
        uri: "./pipeline/SchemaGen/schema/3"
        custom_properties {
          key: "name"
          value {
            string_value: "schema"
          }
        }
        custom_properties {
          key: "producer_component"
          value {
            string_value: "SchemaGen"
          }
        }
        custom_properties {
          key: "state"
          value {
            string_value: "published"
          }
        }
        custom_properties {
          key: "tfx_version"
          value {
            string_value: "1.3.0"
          }
        }
        state: LIVE
        , artifact_type: id: 18
        name: "Schema"
        )]
            additional_properties: {}
            additional_custom_properties: {}
        ))
```

[7]:
```
# Visualize the schema
context.show(schema_gen.outputs['schema'])
```

```
<IPython.core.display.HTML object>
```

Feature name	Type	Presence	Valency	Domain

```
'age'                INT  required                        -
'capital-gain'       INT  required                        -
'capital-loss'       INT  required                        -
'education'       STRING  required              'education'
'education-num'      INT  required                        -
'fnlwgt'             INT  required                        -
'hours-per-week'     INT  required                        -
'label'           STRING  required                  'label'
'marital-status'  STRING  required          'marital-status'
'native-country'  STRING  required          'native-country'
'occupation'      STRING  required              'occupation'
'race'            STRING  required                    'race'
'relationship'    STRING  required            'relationship'
'sex'             STRING  required                     'sex'
'workclass'       STRING  required               'workclass'

                                                                                ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪Values
Domain
'education'       ' 10th', ' 11th', ' 12th', ' 1st-4th', ' 5th-6th', ' 7th-8th',␣
 ↪' 9th', ' Assoc-acdm', ' Assoc-voc', ' Bachelors', ' Doctorate', ' HS-grad', '␣
 ↪Masters', ' Preschool', ' Prof-school', ' Some-college'
'label'           ' <=50K', ' >50K'
'marital-status'  ' Divorced', ' Married-AF-spouse', ' Married-civ-spouse', '␣
 ↪Married-spouse-absent', ' Never-married', ' Separated', ' Widowed'
'native-country'  ' ?', ' Cambodia', ' Canada', ' China', ' Columbia', ' Cuba',␣
 ↪' Dominican-Republic', ' Ecuador', ' El-Salvador', ' England', ' France', '␣
 ↪Germany', ' Greece', ' Guatemala', ' Haiti', ' Honduras', ' Hong', ' Hungary',␣
 ↪' India', ' Iran', ' Ireland', ' Italy', ' Jamaica', ' Japan', ' Laos', '␣
 ↪Mexico', ' Nicaragua', ' Outlying-US(Guam-USVI-etc)', ' Peru', ' Philippines',␣
 ↪' Poland', ' Portugal', ' Puerto-Rico', ' Scotland', ' South', ' Taiwan', '␣
 ↪Thailand', ' Trinadad&Tobago', ' United-States', ' Vietnam', ' Yugoslavia', '␣
 ↪Holand-Netherlands'
'occupation'      ' ?', ' Adm-clerical', ' Armed-Forces', ' Craft-repair', '␣
 ↪Exec-managerial', ' Farming-fishing', ' Handlers-cleaners', '␣
 ↪Machine-op-inspct', ' Other-service', ' Priv-house-serv', ' Prof-specialty', '␣
 ↪Protective-serv', ' Sales', ' Tech-support', ' Transport-moving'
'race'            ' Amer-Indian-Eskimo', ' Asian-Pac-Islander', ' Black', '␣
 ↪Other', ' White'
'relationship'    ' Husband', ' Not-in-family', ' Other-relative', ' Own-child',␣
 ↪' Unmarried', ' Wife'
'sex'             ' Female', ' Male'
```

```
'workclass'        ' ?', ' Federal-gov', ' Local-gov', ' Never-worked', '␣
↪Private', ' Self-emp-inc', ' Self-emp-not-inc', ' State-gov', ' Without-pay'
```

1.2.5 Curating the Schema

Now that you have the inferred schema, you can proceed to revising it to be more robust. For instance, you can restrict the age as you did in Week 1. First, you have to load the `Schema` protocol buffer from the metadata store. You can do this by getting the schema uri from the output of `SchemaGen` then use TFDV's `load_schema_text()` method.

[8]:
```
# Get the schema uri
schema_uri = schema_gen.outputs['schema']._artifacts[0].uri

# Get the schema pbtxt file from the SchemaGen output
schema = tfdv.load_schema_text(os.path.join(schema_uri, 'schema.pbtxt'))
```

With that, you can now make changes to the schema as before. For the purpose of this exercise, you will only modify the age domain but feel free to add more if you want.

[9]:
```
# Restrict the range of the `age` feature
tfdv.set_domain(schema, 'age', schema_pb2.IntDomain(name='age', min=17, max=90))

# Display the modified schema. Notice the `Domain` column of `age`.
tfdv.display_schema(schema)
```

```
                    Type  Presence Valency            Domain
Feature name
'age'               INT     required          min: 17; max: 90
'capital-gain'      INT     required          -
'capital-loss'      INT     required          -
'education'         STRING  required          'education'
'education-num'     INT     required          -
'fnlwgt'            INT     required          -
'hours-per-week'    INT     required          -
'label'             STRING  required          'label'
'marital-status'    STRING  required          'marital-status'
'native-country'    STRING  required          'native-country'
'occupation'        STRING  required          'occupation'
'race'              STRING  required          'race'
'relationship'      STRING  required          'relationship'
'sex'               STRING  required          'sex'
'workclass'         STRING  required          'workclass'
```

```
                                                                                ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪Values
Domain
'education'       ' 10th', ' 11th', ' 12th', ' 1st-4th', ' 5th-6th', ' 7th-8th',␣
 ↪' 9th', ' Assoc-acdm', ' Assoc-voc', ' Bachelors', ' Doctorate', ' HS-grad', '␣
 ↪Masters', ' Preschool', ' Prof-school', ' Some-college'
'label'           ' <=50K', ' >50K'
'marital-status'  ' Divorced', ' Married-AF-spouse', ' Married-civ-spouse', '␣
 ↪Married-spouse-absent', ' Never-married', ' Separated', ' Widowed'
'native-country'  ' ?', ' Cambodia', ' Canada', ' China', ' Columbia', ' Cuba',␣
 ↪' Dominican-Republic', ' Ecuador', ' El-Salvador', ' England', ' France', '␣
 ↪Germany', ' Greece', ' Guatemala', ' Haiti', ' Honduras', ' Hong', ' Hungary',␣
 ↪' India', ' Iran', ' Ireland', ' Italy', ' Jamaica', ' Japan', ' Laos', '␣
 ↪Mexico', ' Nicaragua', ' Outlying-US(Guam-USVI-etc)', ' Peru', ' Philippines',␣
 ↪' Poland', ' Portugal', ' Puerto-Rico', ' Scotland', ' South', ' Taiwan', '␣
 ↪Thailand', ' Trinadad&Tobago', ' United-States', ' Vietnam', ' Yugoslavia', '␣
 ↪Holand-Netherlands'
'occupation'      ' ?', ' Adm-clerical', ' Armed-Forces', ' Craft-repair', '␣
 ↪Exec-managerial', ' Farming-fishing', ' Handlers-cleaners', '␣
 ↪Machine-op-inspct', ' Other-service', ' Priv-house-serv', ' Prof-specialty', '␣
 ↪Protective-serv', ' Sales', ' Tech-support', ' Transport-moving'
'race'            ' Amer-Indian-Eskimo', ' Asian-Pac-Islander', ' Black', '␣
 ↪Other', ' White'
'relationship'    ' Husband', ' Not-in-family', ' Other-relative', ' Own-child',␣
 ↪' Unmarried', ' Wife'
'sex'             ' Female', ' Male'
'workclass'       ' ?', ' Federal-gov', ' Local-gov', ' Never-worked', '␣
 ↪Private', ' Self-emp-inc', ' Self-emp-not-inc', ' State-gov', ' Without-pay'
```

1.2.6 Schema Environments

By default, your schema will watch for all the features declared above including the label. However, when the model is served for inference, it will get datasets that will not have the label because that is the feature that the model will be trying to predict. You need to configure the pipeline to not raise an alarm when this kind of dataset is received.

You can do that with schema environments. First, you will need to declare training and serving environments, then configure the serving schema to not watch for the presence of labels. See how it is implemented below.

[10]:
```
# Create schema environments for training and serving
schema.default_environment.append('TRAINING')
```

```
schema.default_environment.append('SERVING')

# Omit label from the serving environment
tfdv.get_feature(schema, 'label').not_in_environment.append('SERVING')
```

You can now freeze the curated schema and save to a local directory.

[11]:
```
# Declare the path to the updated schema directory
_updated_schema_dir = f'{_pipeline_root}/updated_schema'

# Create the said directory
!mkdir -p {_updated_schema_dir}

# Declare the path to the schema file
schema_file = os.path.join(_updated_schema_dir, 'schema.pbtxt')

# Save the curated schema to the said file
tfdv.write_schema_text(schema, schema_file)
```

1.2.7 ImportSchemaGen

Now that the schema has been saved, you need to create an artifact in the metadata store that will point to it. TFX provides the ImporterSchemaGen component used to import a curated schema to ML Metadata. You simply need to specify the URL of the revised schema file.

[12]:
```
# Use ImportSchemaGen to put the curated schema to ML Metadata
user_schema_importer = tfx.components.ImportSchemaGen(schema_file=schema_file)

# Run the component
context.run(user_schema_importer, enable_cache=False)
```

[12]:
```
ExecutionResult(
    component_id: ImportSchemaGen
    execution_id: 4
    outputs:
        schema: Channel(
            type_name: Schema
            artifacts: [Artifact(artifact: id: 4
        type_id: 18
        uri: "./pipeline/ImportSchemaGen/schema/4"
        custom_properties {
          key: "name"
          value {
            string_value: "schema"
          }
        }
        custom_properties {
```

```
    key: "producer_component"
    value {
      string_value: "ImportSchemaGen"
    }
  }
  custom_properties {
    key: "state"
    value {
      string_value: "published"
    }
  }
  custom_properties {
    key: "tfx_version"
    value {
      string_value: "1.3.0"
    }
  }
  state: LIVE
  , artifact_type: id: 18
  name: "Schema"
  )]
      additional_properties: {}
      additional_custom_properties: {}
  ))
```

If you pass in the component output to `context.show()`, then you should see the schema.

[13]:
```
# See the result
context.show(user_schema_importer.outputs['schema'])
```

```
<IPython.core.display.HTML object>
```

Feature name	Type	Presence	Valency	Domain
'age'	INT	required		min: 17; max: 90
'capital-gain'	INT	required		-
'capital-loss'	INT	required		-
'education'	STRING	required		'education'
'education-num'	INT	required		-
'fnlwgt'	INT	required		-
'hours-per-week'	INT	required		-
'label'	STRING	required		'label'
'marital-status'	STRING	required		'marital-status'
'native-country'	STRING	required		'native-country'
'occupation'	STRING	required		'occupation'
'race'	STRING	required		'race'
'relationship'	STRING	required		'relationship'
'sex'	STRING	required		'sex'

```
'workclass'       STRING  required          'workclass'

                                                                                ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪                                                                              ␣
 ↪Values
Domain
'education'       ' 10th', ' 11th', ' 12th', ' 1st-4th', ' 5th-6th', ' 7th-8th',␣
 ↪' 9th', ' Assoc-acdm', ' Assoc-voc', ' Bachelors', ' Doctorate', ' HS-grad', '␣
 ↪Masters', ' Preschool', ' Prof-school', ' Some-college'
'label'           ' <=50K', ' >50K'
'marital-status'  ' Divorced', ' Married-AF-spouse', ' Married-civ-spouse', '␣
 ↪Married-spouse-absent', ' Never-married', ' Separated', ' Widowed'
'native-country'  ' ?', ' Cambodia', ' Canada', ' China', ' Columbia', ' Cuba',␣
 ↪' Dominican-Republic', ' Ecuador', ' El-Salvador', ' England', ' France', '␣
 ↪Germany', ' Greece', ' Guatemala', ' Haiti', ' Honduras', ' Hong', ' Hungary',␣
 ↪' India', ' Iran', ' Ireland', ' Italy', ' Jamaica', ' Japan', ' Laos', '␣
 ↪Mexico', ' Nicaragua', ' Outlying-US(Guam-USVI-etc)', ' Peru', ' Philippines',␣
 ↪' Poland', ' Portugal', ' Puerto-Rico', ' Scotland', ' South', ' Taiwan', '␣
 ↪Thailand', ' Trinadad&Tobago', ' United-States', ' Vietnam', ' Yugoslavia', '␣
 ↪Holand-Netherlands'
'occupation'      ' ?', ' Adm-clerical', ' Armed-Forces', ' Craft-repair', '␣
 ↪Exec-managerial', ' Farming-fishing', ' Handlers-cleaners', '␣
 ↪Machine-op-inspct', ' Other-service', ' Priv-house-serv', ' Prof-specialty', '␣
 ↪Protective-serv', ' Sales', ' Tech-support', ' Transport-moving'
'race'            ' Amer-Indian-Eskimo', ' Asian-Pac-Islander', ' Black', '␣
 ↪Other', ' White'
'relationship'    ' Husband', ' Not-in-family', ' Other-relative', ' Own-child',␣
 ↪' Unmarried', ' Wife'
'sex'             ' Female', ' Male'
'workclass'       ' ?', ' Federal-gov', ' Local-gov', ' Never-worked', '␣
 ↪Private', ' Self-emp-inc', ' Self-emp-not-inc', ' State-gov', ' Without-pay'
```

1.2.8 ExampleValidator

You can then use this new artifact as input to the other components of the pipeline. See how it is used as the `schema` argument in `ExampleValidator` below.

[14]:
```
# Instantiate ExampleValidator with the StatisticsGen and SchemaGen ingested␣
 ↪data
example_validator = tfx.components.ExampleValidator(
    statistics=statistics_gen.outputs['statistics'],
    schema=user_schema_importer.outputs['schema'])
```

```
# Run the component.
context.run(example_validator)
```

[14]:
```
ExecutionResult(
    component_id: ExampleValidator
    execution_id: 5
    outputs:
        anomalies: Channel(
            type_name: ExampleAnomalies
            artifacts: [Artifact(artifact: id: 5
        type_id: 21
        uri: "./pipeline/ExampleValidator/anomalies/5"
        properties {
          key: "split_names"
          value {
            string_value: "[\"train\", \"eval\"]"
          }
        }
        custom_properties {
          key: "name"
          value {
            string_value: "anomalies"
          }
        }
        custom_properties {
          key: "producer_component"
          value {
            string_value: "ExampleValidator"
          }
        }
        custom_properties {
          key: "state"
          value {
            string_value: "published"
          }
        }
        custom_properties {
          key: "tfx_version"
          value {
            string_value: "1.3.0"
          }
        }
        state: LIVE
        , artifact_type: id: 21
        name: "ExampleAnomalies"
        properties {
          key: "span"
```

```
    value: INT
  }
  properties {
    key: "split_names"
    value: STRING
  }
  )]
      additional_properties: {}
      additional_custom_properties: {}
  ))
```

[15]:
```
# Visualize the results
context.show(example_validator.outputs['anomalies'])
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

1.2.9 Practice with ML Metadata

At this point, you should now take some time exploring the contents of the metadata store saved by your component runs. This will let you practice tracking artifacts and how they are related to each other. This involves looking at artifacts, executions, and events. This skill will let you recover related artifacts even without seeing the code of the training run. All you need is access to the metadata store.

See how the input artifact IDs to an instance of `ExampleAnomalies` are tracked in the following cells. If you have this notebook, then you will already know that it uses the output of StatisticsGen for this run and also the curated schema you imported. However, if you already have hundreds of training runs and parameter iterations, you may find it hard to track which is which. That's where the metadata store will be useful. Since it records information about a specific pipeline run, you will be able to track the inputs and outputs of a particular artifact.

You will start by setting the connection config to the metadata store.

[16]:
```
# Import mlmd and utilities
import ml_metadata as mlmd
from ml_metadata.proto import metadata_store_pb2

# Get the connection config to connect to the context's metadata store
connection_config = context.metadata_connection_config

# Instantiate a MetadataStore instance with the connection config
store = mlmd.MetadataStore(connection_config)
```

Next, let's see what artifact types are available in the metadata store.

[17]:
```
# Get artifact types
artifact_types = store.get_artifact_types()

# Print the results
[artifact_type.name for artifact_type in artifact_types]
```

[17]:
```
['Examples', 'ExampleStatistics', 'Schema', 'ExampleAnomalies']
```

If you get the artifacts of type `Schema`, you will see that there are two entries. One is the inferred and the other is the one you imported. At the end of this exercise, you can verify that the curated schema is the one used for the `ExampleValidator` run we will be investigating.

[18]:
```
# Get artifact types
schema_list = store.get_artifacts_by_type('Schema')

[(f'schema uri: {schema.uri}', f'schema id:{schema.id}') for schema in␣
 ↪schema_list]
```

[18]:
```
[('schema uri: ./pipeline/SchemaGen/schema/3', 'schema id:3'),
 ('schema uri: ./pipeline/ImportSchemaGen/schema/4', 'schema id:4')]
```

Let's get the first instance of `ExampleAnomalies` to get the output of `ExampleValidator`.

[19]:
```
# Get 1st instance of ExampleAnomalies
example_anomalies = store.get_artifacts_by_type('ExampleAnomalies')[0]

# Print the artifact id
print(f'Artifact id: {example_anomalies.id}')
```

```
Artifact id: 5
```

You will use the artifact ID to get events related to it. Let's just get the first instance.

[20]:
```
# Get first event related to the ID
anomalies_id_event = store.get_events_by_artifact_ids([example_anomalies.id])[0]

# Print results
print(anomalies_id_event)
```

```
artifact_id: 5
execution_id: 5
path {
  steps {
    key: "anomalies"
  }
  steps {
    index: 0
```

```
  }
}
type: OUTPUT
milliseconds_since_epoch: 1661018287202
```

As expected, the event type will be an `OUTPUT` because this is the output of the `ExampleValidator` component. Since we want to get the inputs, we can track it through the execution id.

[21]:
```
# Get execution ID
anomalies_execution_id = anomalies_id_event.execution_id

# Get events by the execution ID
events_execution = store.get_events_by_execution_ids([anomalies_execution_id])

# Print results
print(events_execution)
```

```
[artifact_id: 2
execution_id: 5
path {
  steps {
    key: "statistics"
  }
  steps {
    index: 0
  }
}
type: INPUT
milliseconds_since_epoch: 1661018286855
, artifact_id: 4
execution_id: 5
path {
  steps {
    key: "schema"
  }
  steps {
    index: 0
  }
}
type: INPUT
milliseconds_since_epoch: 1661018286855
, artifact_id: 5
execution_id: 5
path {
  steps {
    key: "anomalies"
  }
```

```
  steps {
    index: 0
  }
}
type: OUTPUT
milliseconds_since_epoch: 1661018287202
]
```

We see the artifacts which are marked as `INPUT` above representing the statistics and schema inputs. We can extract their IDs programmatically like this. You will see that you will get the artifact ID of the curated schema you printed out earlier.

[22]:
```
# Filter INPUT type events
inputs_to_exval = [event.artifact_id for event in events_execution
                       if event.type == metadata_store_pb2.Event.INPUT]

# Print results
print(inputs_to_exval)
```

```
[2, 4]
```

Congratulations! You have now completed this notebook on iterative schemas and saw how it can be used in a TFX pipeline. You were also able to track an artifact's lineage by looking at the artifacts, events, and executions in the metadata store. These will come in handy in this week's assignment!