

Type-safe Machine Learning Orchestration with Flyte and Pandera



Data Council Austin 2022

Niels Bantilan, ML Engineer @ Union.ai

03/23/2022

Type-safety is a critical feature of *orchestration tools* that deal with *data* and *machine learning*

Types define the *set of values* that data can take, but they also define the *domain of operations* that we can perform on that data.

integers $\in \{ 1, 2, -1, 5, 1000, \dots \}$

strings $\in \{ "a", "xyz", "hello", "foobar", \dots \}$

✓ $1 + 1 \rightarrow 2$

✗ $1 + "a" \rightarrow \text{undefined}$

✓ $\text{mean}([1, 2, 3]) \rightarrow 2$

✗ $\text{mean}(["a", "b", "a", "c"]) \rightarrow \text{undefined}$

Types can be simple:

```
int, float, str
```

Or more complex:

```
list[int]
```

```
dict[str, float]
```

```
dict[str, list[float]]
```

Let's talk about housing

7.2.7. California Housing dataset

Data Set Characteristics:

Number of Instances:	20640
Number of Attributes:	8 numeric, predictive attributes and the target
Attribute Information:	<ul style="list-style-type: none">• MedInc median income in block group• HouseAge median house age in block group• AveRooms average number of rooms per household• AveBedrms average number of bedrooms per household• Population block group population• AveOccup average number of household members• Latitude block group latitude• Longitude block group longitude
Missing Attribute Values:	None

Source: https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset

Let's talk about housing 🏠

```
pandas.DataFrame({  
    'Latitude': [37.88, ...],  
    'Longitude': [-122.23, ...],  
    'AveBedrms': [1.0238, ...],  
    'AveOccup': [2.5555, ...],  
    'AveRooms': [6.9841, ...],  
    'HouseAge': [41.0, ...],  
    'MedInc': [8.3252, ...],  
    'Population': [322.0, ...],  
    'MedHouseVal': [4.526, ...],  
})
```

} float

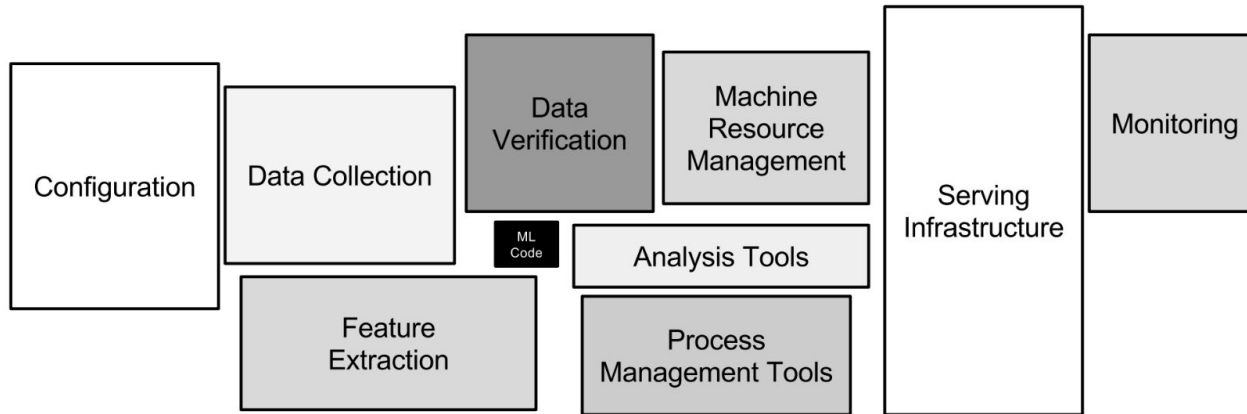
} positive float

Source: https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

Enforcing and **maintaining** data quality is challenging

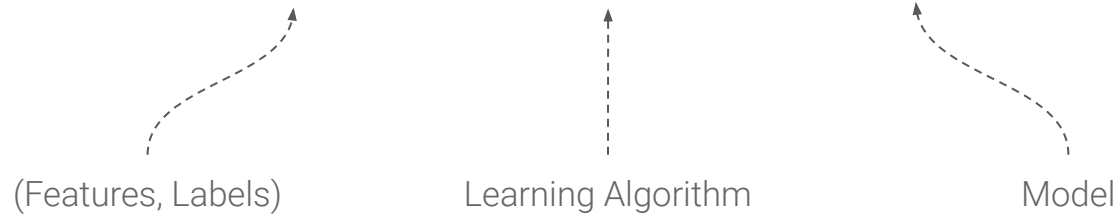
Production machine learning has a *complexity* problem

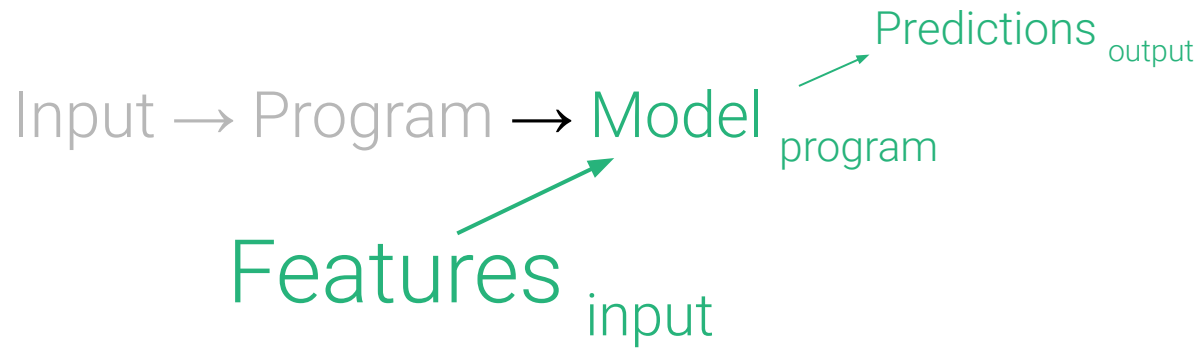
How do I know if these components are compatible?



source: <https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcdf2674f757a2463eba-Paper.pdf>

Input → Program → Output







Strongly-typed interfaces unlock static analysis capabilities that push many potential errors from the *runtime context* into the *compile-time context*.


Reliability

Readability: as a human being  or machine , I can tell what a component needs as input and what it produces as output.

Reproducibility: when a component fails  at its input/output boundaries, I can be more confident that I can reproduce the error .



Efficiency

Caching: if I want to determine whether I should hit the cache  or re-compute  the result of a component, I can first check for changes in a function's type signature before checking actual input values.

Parallelization: before I try to concurrently apply functions to a collection of inputs , I can be confident that the elements in the collection are of the correct type.


Auditability

Debugging: When a pipeline execution fails , I can pinpoint the cause of the error quickly and understand how to address it.

Data Lineage: I can understand how some downstream artifact  came to be by looking at the upstream processes  that produced it.

Flyte is a *data-* and *machine-learning-*aware **orchestration tool** with **type-safety** built into multiple layers of the software stack.

Flyte

Easily Compose
Workflows 
using Tasks as
Building Blocks



```
pip install flytekit
```

```
from flytekit import task, workflow

@task
def get_data(): ...

@task
def process_data(): ...

@task
def train_model(): ...

@workflow
def training_workflow():
    data = get_data()
    processed_data = process_data(data=data)
    return train_model(processed_data=processed_data)
```

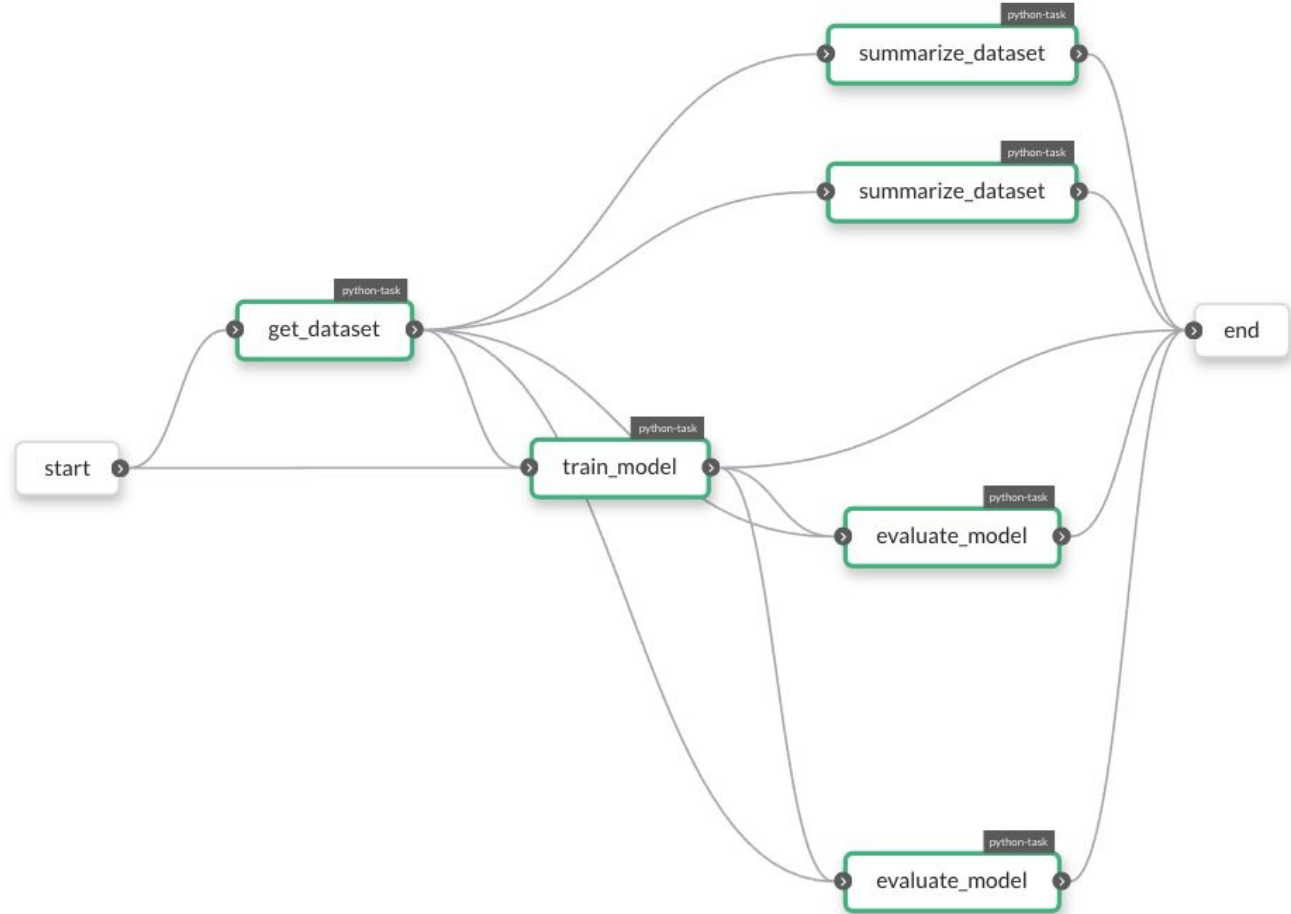
California House Price Regression

```
pandas.DataFrame({  
    'Latitude': [37.88, ...],  
    'Longitude': [-122.23, ...],  
    'AveBedrms': [1.0238, ...],  
    'AveOccup': [2.5555, ...],  
    'AveRooms': [6.9841, ...],  
    'HouseAge': [41.0, ...],  
    'MedInc': [8.3252, ...],  
    'Population': [322.0, ...],  
    'MedHouseVal': [4.526, ...],  
})
```

features

target

Pipeline Overview



What Types are
We Going to
Use?



```
Dataset = Annotated[
    pd.DataFrame,
    kwtypes(
        Latitude=float,
        Longitude=float,
        AveBedrms=float,
        AveOccup=float,
        AveRooms=float,
        HouseAge=float,
        MedInc=float,
        MedHouseVal=float,
    )
]

TARGET = "MedHouseVal"

DatasetSplits = NamedTuple(
    "DatasetSplits", train=Dataset, test=Dataset
)

TrainingResult = NamedTuple(
    "TrainingResult", model=Ridge, train_mse=float, test_mse=float
)

You, now | 1 author (You)
@dataclass_json
@dataclass
class Hyperparameters:
    alpha: float
    random_state: int = 42
```

Tasks are 
Containerized
Units of Work
 with a
Transparent
Interface

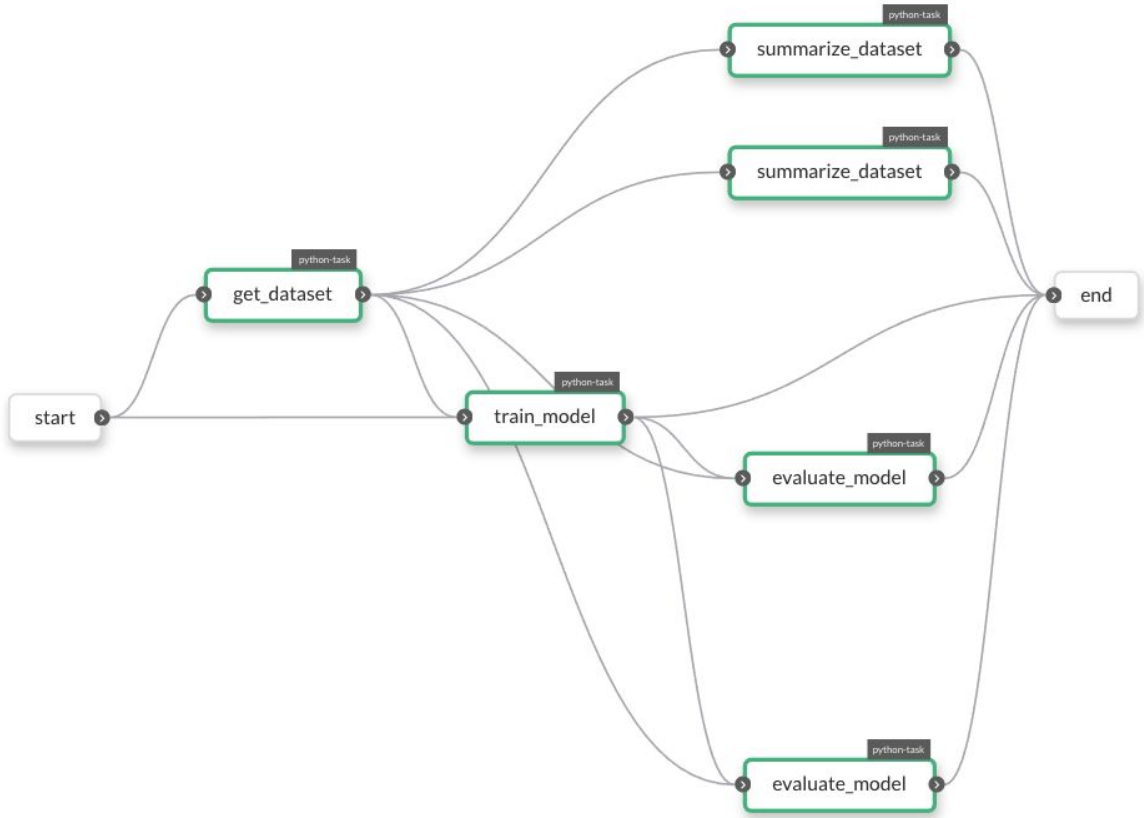
```
@task
def get_dataset(test_size: float, random_state: int) -> DatasetSplits:
    dataset = fetch_california_housing(as_frame=True).frame
    return train_test_split(dataset, test_size=test_size, random_state=random_state)

@task
def summarize_dataset(dataset: Dataset) -> pd.DataFrame:
    return dataset.describe()

@task
def train_model(dataset: Dataset, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])

@task
def evaluate_model(dataset: Dataset, model: Ridge) -> float:
    features, target = dataset.drop(TARGET, axis="columns"), dataset[TARGET]
    return mean_squared_error(target, model.predict(features))
```

Workflows are
Dynamic DAGs
that Compose
Tasks Together
to do Something
Useful 🏗️



Auto-generate Strongly Typed Launch Forms



Workflow execution interface for `california_housing_regression.simple_workflows.main`.

Create New Execution
`california_housing_regression.simple_workflows.main`

Workflow Version: `bdc0462083afb8861d436d28301efc4efde35402`

Launch Plan: `california_housing_regression.simple_workflows.main`

Inputs
Enter input values below. Items marked with an asterisk(*) are required.

hyperparameters (struct)*

- `random_state (integer)`: 42
- `alpha (float)`
- `random_state (integer)`: 43
- `random_state`
- `test_size (float)`: 0.2
- `test_size`

Buttons: **Cancel** | **Launch**

TIME CREATED	STATUS	START TIME
3/21/2022 3:34:04 PM UTC		
	SUCCEEDED	3/21/2022 3:41:00 3/21/2022 11:41:00

Docker

Guarantees Reproducibility

...as long as tasks are
idempotent

```
FROM python:3.9-slim-buster
```

```
WORKDIR /root
```

```
ENV VENV /opt/venv
```

```
ENV LANG C.UTF-8
```

```
ENV LC_ALL C.UTF-8
```

```
ENV PYTHONPATH /root
```

```
# e.g. flyte.config or sandbox.config
```

```
ARG config
```

```
RUN apt-get update && \  
    apt-get install -y \  
        libsm6 \  
        libxext6 \  
        libxrender-dev \  
        ffmpeg \  
        build-essential
```

```
# Install the AWS cli separately to prevent issues with boto being written over
```

```
RUN pip3 install awscli
```

```
ENV VENV /opt/venv
```

```
# Virtual environment
```

```
RUN python3 -m venv ${VENV}
```

```
ENV PATH="${VENV}/bin:$PATH"
```

```
# Install Python dependencies
```

```
COPY requirements.txt /root
```

```
RUN pip install -r /root/requirements.txt
```

```
COPY california_housing_regression /root/california_housing_regression
```

```
COPY $config /root/flyte.config
```

```
# This image is supplied by the build script and will be used to determine the version
```

```
# when registering tasks, workflows, and launch plans
```

```
ARG image
```

```
ENV FLYTE_INTERNAL_IMAGE $image
```

Flyte Statically Analyzes the DAG to catch Type Errors

```
@task
def train_model(dataset: Dataset, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])
```



```
@task
def train_model_type_error(dataset: dict, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])

# TypeError: Cannot convert from scalar {
#   schema {
#     uri: "/tmp/flyte/20220319_170441/raw/f6608163de0159a39b9d21456bf4dc17"
#     type {
#       columns {name: "Latitude" type: FLOAT}
#       columns {name: "Longitude" type: FLOAT}
#       columns {name: "AveBedrms" type: FLOAT}
#       columns {name: "AveOccup" type: FLOAT}
#       columns {name: "AveRooms" type: FLOAT}
#       columns {name: "HouseAge" type: FLOAT}
#       columns {name: "MedInc" type: FLOAT}
#       columns {name: "MedHouseVal" type: FLOAT}
#     }
#   }
# }
# to <class 'dict'>
```

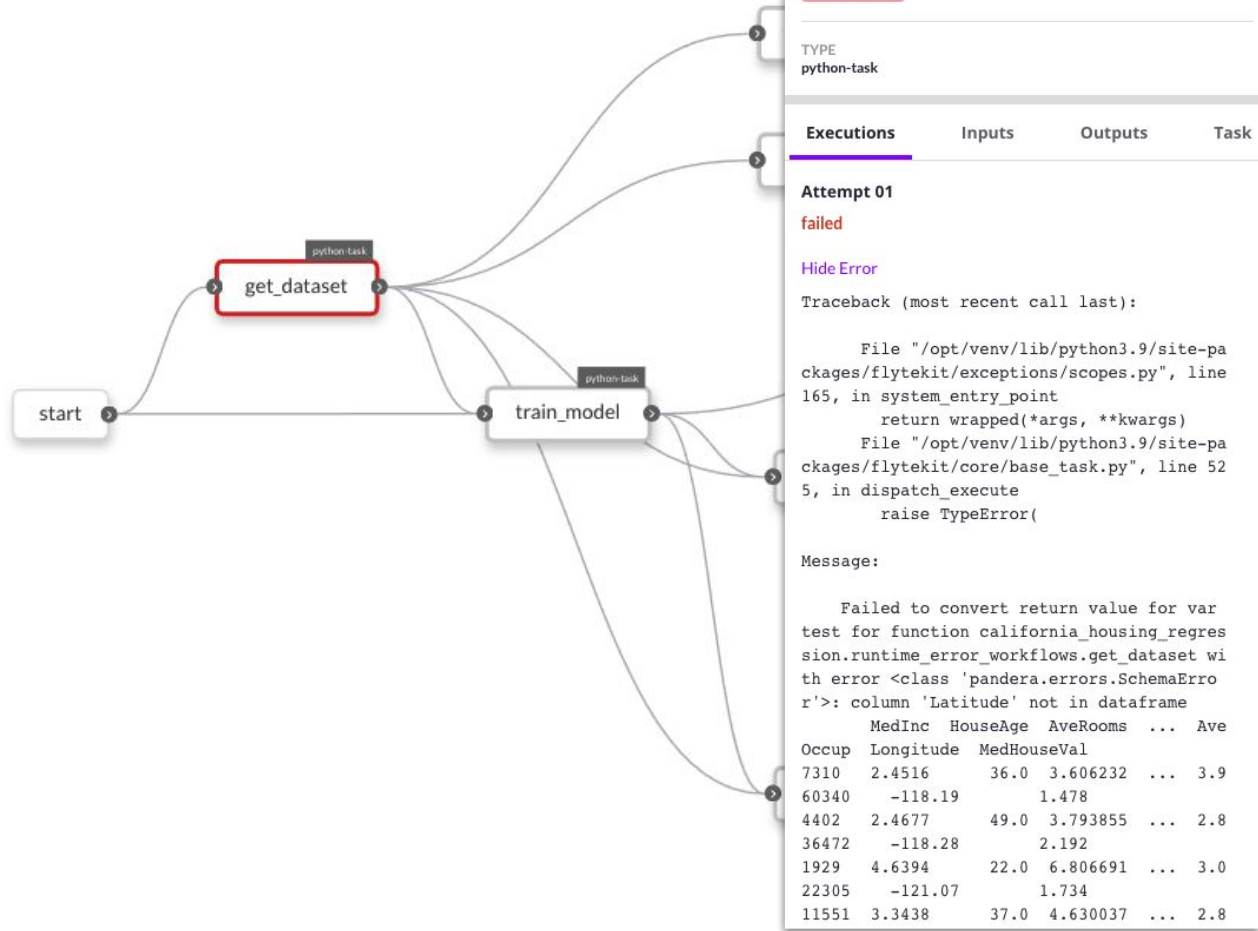
Catch Value Errors 🐞 When Testing Locally

```
@task
def get_dataset(test_size: float, random_state: int) -> DatasetSplits:
    dataset = fetch_california_housing(as_frame=True).frame
    training_set, test_set = train_test_split(
        dataset, test_size=test_size, random_state=random_state
    )
    # corrupt the test set
    test_set = test_set.drop("Latitude", axis="columns")
    return training_set, test_set
```

TypeError: Failed to convert return value for var test for function __main__.get_dataset with error <class 'pandera.errors.SchemaError': column 'Latitude' not in dataframe

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Longitude	MedHouseVal
7310	2.4516	36.0	3.606232	1.073654	1398.0	3.960340	-118.19	1.478
4402	2.4677	49.0	3.793855	1.186323	2862.0	2.836472	-118.28	2.192
1929	4.6394	22.0	6.806691	1.018587	813.0	3.022305	-121.07	1.734
11551	3.3438	37.0	4.630037	1.003663	783.0	2.868132	-117.98	1.996
9882	3.0608	22.0	4.750515	1.039863	3794.0	2.607560	-121.79	1.683

Know Where your Pipeline Blew Up 🧨



Cache the Outputs of a Task

```
@task(cache=True, cache_version="1.0")
def get_dataset(test_size: float, random_state: int) -> DatasetSplits:
    dataset = fetch_california_housing(as_frame=True).frame
    return train_test_split(dataset, test_size=test_size, random_state=random_state)
```

```
@task(cache=True, cache_version="1.0")
def summarize_dataset(dataset: Dataset) -> pd.DataFrame:
    return dataset.describe()
```

```
@task(cache=True, cache_version="1.0")
def train_model(dataset: Dataset, hyperparameters: Hyperparameters) -> Ridge:
    model = Ridge(**asdict(hyperparameters))
    return model.fit(dataset.drop(TARGET, axis="columns"), dataset[TARGET])
```

```
@task(cache=True, cache_version="1.0")
def evaluate_model(dataset: Dataset, model: Ridge) -> float:
    features, target = dataset.drop(TARGET, axis="columns"), dataset[TARGET]
    # corrupt the features
    features = features.drop("Latitude", axis="columns")
    return mean_squared_error(target, model.predict(features))
```


Don't
Re-compute,
Hit the Cache!



```
@task(cache=True, cache_version="1.0")
def evaluate_model(dataset: Dataset, model: Ridge) -> float:
    features, target = dataset.drop(TARGET, axis="columns"), dataset[TARGET]
    # corrupt the features
    features = features.drop("Latitude", axis="columns")
    return mean_squared_error(target, model.predict(features))
```

View Inputs & Outputs

Recover

Relaunch

get_dataset california_housing_regression	n0	Python-Task	SUCCEEDED	↺
summarize_dataset california_housing_regression	n1	Python-Task	SUCCEEDED	↺
summarize_dataset california_housing_regression	n2	Python-Task	SUCCEEDED	↺
train_model california_housing_regression	n3	Python-Task	SUCCEEDED	↺

Workflows Execute Tasks with Built-in Parallelism

```
@workflow
def main(
    hyperparameters: Hyperparameters,
    test_size: float = 0.2,
    random_state: int = 43,
) -> TrainingResult:
    train_dataset, test_dataset = get_dataset(
        test_size=test_size, random_state=random_state
    )

    summarize_dataset(dataset=train_dataset)
    summarize_dataset(dataset=test_dataset)

    model = train_model(dataset=train_dataset, hyperparameters=hyperparameters)
    train_mse = evaluate_model(dataset=train_dataset, model=model)
    test_mse = evaluate_model(dataset=test_dataset, model=model)

    return model, train_mse, test_mse
```


Static Type Checking Applies to Parallelized Invocations of a Task

```
@task
def summarize_dataset(dataset: Dataset) -> pd.DataFrame:
    return dataset.describe()
```

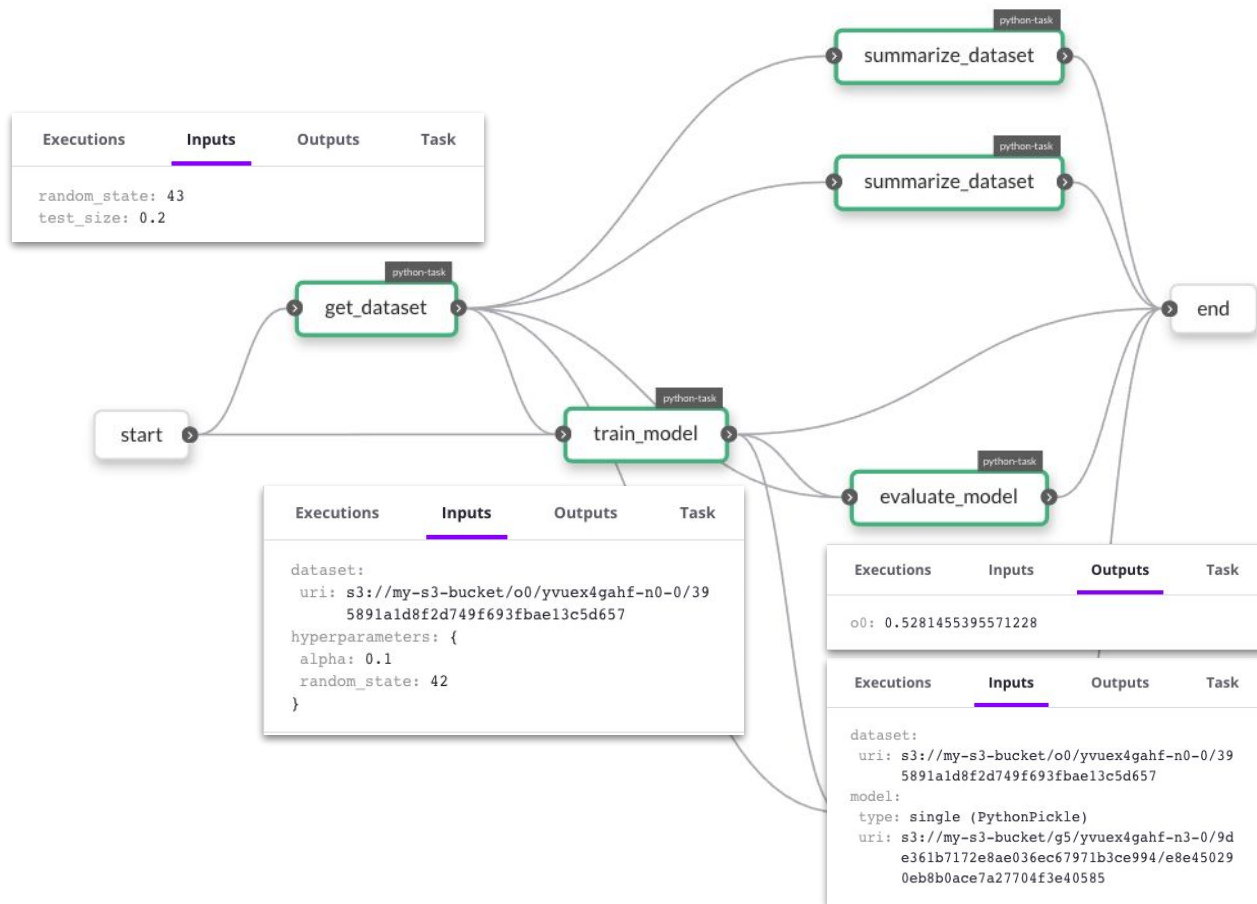
```
@task
def summarize_dataset(dataset: dict) -> pd.DataFrame:
    return dataset.describe()
```

```
@workflow
def main(
    hyperparameters: Hyperparameters,
    test_size: float = 0.2,
    random_state: int = 43,
) -> TrainingResult:
    train_dataset, test_dataset = get_dataset(test_size=test_size, random_state=random_state)
    summarize_dataset(dataset=train_dataset)
    summarize_dataset(dataset=test_dataset)

    ...
```

```
TypeError: Cannot convert from scalar {
  schema {
    uri: "/tmp/flyte/20220321_133605/raw/abe1d6d3bf9e88288a5ce4d1e1d44b55"
    type {
    }
  }
}
to <class 'dict'>
```

Trace Model Artifacts to the Data and Downstream Processes that Produced it



But wait, what about **data types** for *machine learning*?

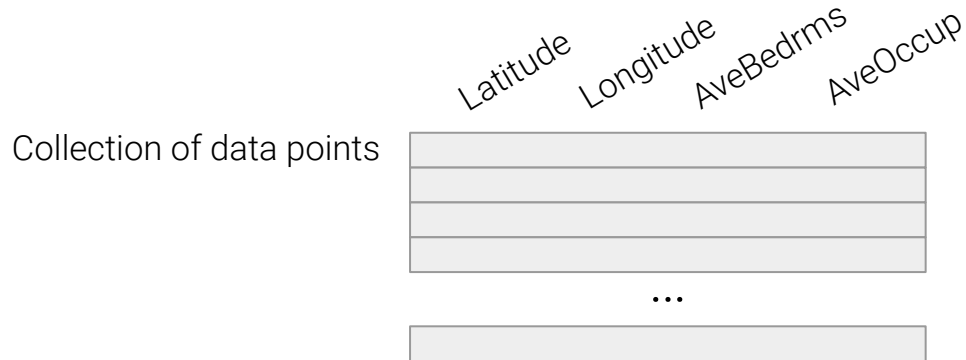
Pandera is a **statistical typing** and **data testing** library for *dataframes*, providing tools for defining *complex data types* and *unit testing* your pipelines with them.

Statistical Typing: Specifying the properties of collections of data points



- Primitive data types
- Value range
- Allowable values
- Regex string match
- Nullability

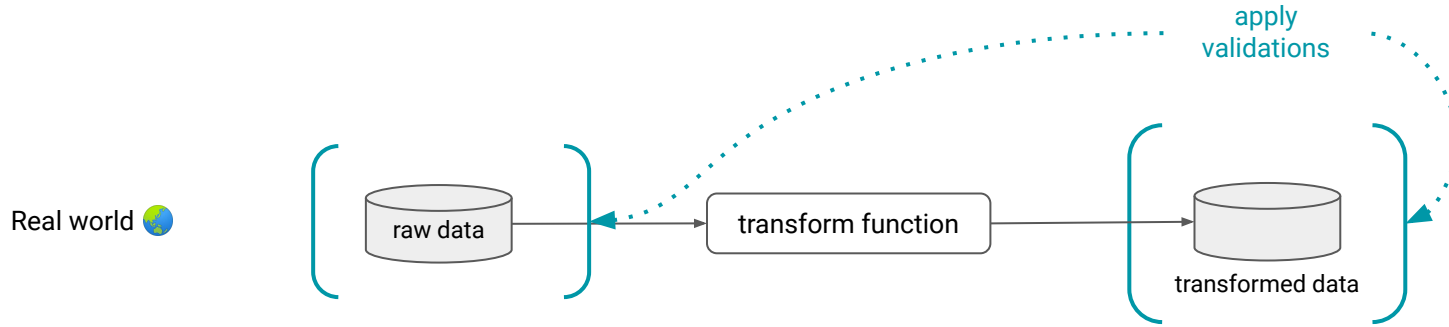
Statistical Typing: Specifying the properties of collections of data points



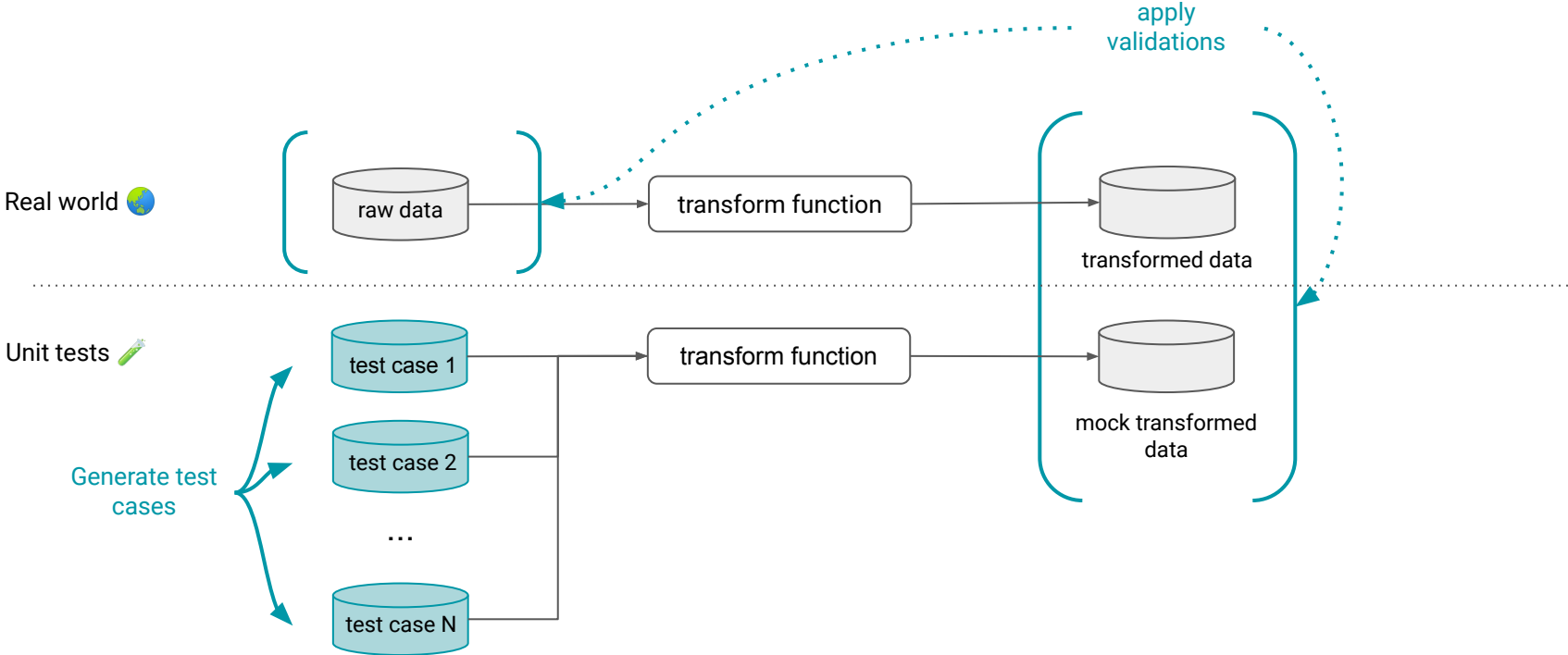
- Apply atomic checks at scale
- Uniqueness
- Monotonicity
- Mean, median, standard deviation
- Statistical distributions
- Fractional checks, e.g. “90% of data points are not null”

Statistical properties, by definition, can only be verified at *runtime*, but we can also define *functions* that use **statistical type annotations** that verify valid operations on those types.



Data Testing: Validating not only real data...



... but also the functions that produce them



Pandera

Define
Statistical
Types for your
DataFrame-like
Objects  

```
pip install pandera
```

```
import pandera as pa
from pandera.typing import Series, DataFrame

class MySchema(pa.SchemaModel):
    col1: Series[float]
    col2: Series[int]
    col3: Series[str]

@pa.check_types
def func(df: DataFrame[MySchema]):
    ...
```

Pandera and Flyte Play Well Together 🤝

```
pip install flytekitplugins-pandera
```

```
import flytekitplugins.pandera
import pandera as pa
from flytekit import task
from pandera.typing import Series, DataFrame

class MySchema(pa.SchemaModel):
    col1: Series[float]
    col2: Series[int]
    col3: Series[str]

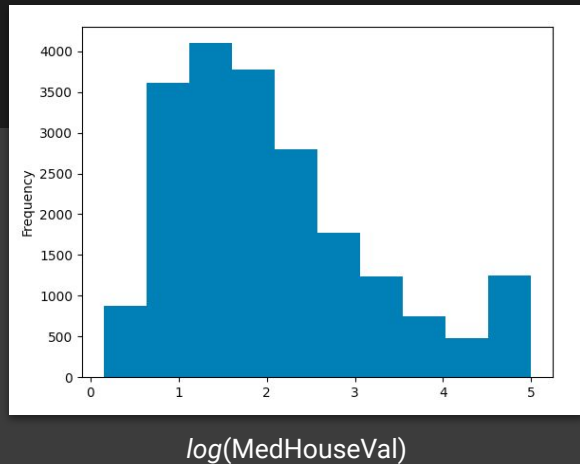
@task
def func(df: DataFrame[MySchema]):
    ...
```

Defining a Statistical Type for California Housing Dataset



```
class CaliforniaHousingData(pa.SchemaModel):
    Latitude: Series[float] = pa.Field(in_range={"min_value": -90, "max_value": 90})
    Longitude: Series[float] = pa.Field(in_range={"min_value": -180, "max_value": 180})
    AveBedrms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveOccup: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveRooms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    HouseAge: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedInc: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedHouseVal: Series[float] = pa.Field(
        mean_eq={
            "value": 2.0685,
            "alpha": 1e-3,
            "error": "MedHouseVal mean value is not equal to 2.0685 [alpha=1e-3]",
        }
    )
```

```
class Config:
    coerce = True
```



Custom Checks are Just...

 Functions 

```
def mean_eq(pandas_obj, *, value, alpha):
    """
    Null hypothesis: the mean of data is equal to the value argument.
    If pvalue is greater than alpha, we can't reject the null hypothesis
    """
    _, pvalue = stats.ttest_1samp(pandas_obj, value)
    return pvalue >= alpha

def mean_eq_strategy(
    pandera_dtype: pa.DataType,
    strategy: Optional[st.SearchStrategy] = None,
    *,
    value,
    alpha,
):
    if strategy:
        raise pa.errors.BaseStrategyOnlyError(
            "mean_eq_strategy is a base strategy. You cannot specify the "
            "strategy argument to chain it to a parent strategy."
        )
    return pandas_dtype_strategy(
        pandera_dtype,
        strategy=st.builds(lambda: np.random.normal(loc=value, scale=0.01))
    )

extensions.register_check_method(
    mean_eq,
    statistics=["value", "alpha"],
    strategy=mean_eq_strategy,
    supported_types=[pd.Series],
    check_type="vectorized",
)
```

Know When Your Data Has Missing Columns



TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:45:38 PM UTC 3/21/2022 2:45:38 PM EDT	40s	View Logs

```
[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last):  
  
File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_point  
    return wrapped(*args, **kwargs)  
File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute  
    raise TypeError(  
  
Message:  
  
Failed to convert return value for var test for function california_housing_regression.pandera_column_error_workflows.get_dataset with error <class 'pandera.errors.SchemaError': column 'Latitude' not in dataframe  
MedInc HouseAge AveRooms ... AveOccup Longitude MedHouseVal  
7310 2.4516 36.0 3.606232 ... 3.960340 -118.19 1.478  
4402 2.4677 49.0 3.793855 ... 2.836472 -118.28 2.192  
1929 4.6394 22.0 6.806691 ... 3.022305 -121.07 1.734  
11551 3.3438 37.0 4.630037 ... 2.868132 -117.98 1.996  
9882 3.0608 22.0 4.750515 ... 2.607560 -121.79 1.683  
  
[5 rows x 8 columns]
```

Know When Your Data Has the Wrong Type

TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:51:21 PM UTC 3/21/2022 2:51:21 PM EDT	44s	View Logs
<pre>[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last): File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_point return wrapped(*args, **kwargs) File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute raise TypeError(Message: Failed to convert return value for var test for function california_housing_regression.pandera_dtype_error_workflows.get_dataset with error <class 'pandera.errors.SchemaError'>: Error while coercing 'Latitude' to type float64: Could not coerce <class 'pandas.core.series.Series'> data_container into type float64: index failure_case 0 7310 N/A 1 4402 N/A 2 1929 N/A 3 11551 N/A 4 9882 N/A</pre>						

Know When Your Data Has the Wrong Values



TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:52:01 PM UTC 3/21/2022 2:52:01 PM EDT	45s	View Logs

```
[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last):  
  
File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_point  
    return wrapped(*args, **kwargs)  
File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute  
    raise TypeError(  
  
Message:  
  
Failed to convert return value for var test for function california_housing_regression.pandera_value_error_workflows.get_dataset with error <class 'pandera.errors.SchemaError': <Schema Column(name=Latitude, type=DataType(float64))> failed element-wise validator 0:  
<Check in_range: in_range(-90, 90)>  
failure cases:  
  index  failure_case  
0  7310      -1000.0  
1  4402      -1000.0  
2  1929      -1000.0  
3  11551     -1000.0  
4   9882     -1000.0
```


Know When Your Data Has the Wrong Statistical Distribution



TASK NAME	NODE ID	TYPE	STATUS	START TIME	DURATION Queued Time	LOGS
get_dataset california_housing_regression	n0	Python-Task	FAILED	3/21/2022 6:52:04 PM UTC 3/21/2022 2:52:04 PM EDT	45s	View Logs

```
[3/3] currentAttempt done. Last Error: SYSTEM::Traceback (most recent call last):  
  
File "/opt/venv/lib/python3.9/site-packages/flytekit/exceptions/scopes.py", line 165, in system_entry_point  
    return wrapped(*args, **kwargs)  
File "/opt/venv/lib/python3.9/site-packages/flytekit/core/base_task.py", line 525, in dispatch_execute  
    raise TypeError(  
  
Message:  
  
Failed to convert return value for var test for function california_housing_regression.pandera_stats_error_workflows.get_dataset with error <class 'pandera.errors.SchemaError'>: <Schema Column(name=MedHouseVal, type=DataTypes(float64))> failed series or dataframe validator 0:  
<Check mean_eq: MedHouseVal mean value is not equal to 2.0685 [alpha=1e-3]>
```

Synthesize Valid Data Under Your Schema's Constraints 🧠

```
class CaliforniaHousingData(pa.SchemaModel):
    Latitude: Series[float] = pa.Field(in_range={"min_value": -90, "max_value": 90})
    Longitude: Series[float] = pa.Field(in_range={"min_value": -180, "max_value": 180})
    AveBedrms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveOccup: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    AveRooms: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    HouseAge: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedInc: Series[float] = pa.Field(in_range={"min_value": 0, "max_value": 1_000_000})
    MedHouseVal: Series[float] = pa.Field(
        mean_eq={
            "value": 2.0685,
            "alpha": 1e-3,
            "error": "MedHouseVal mean value is not equal to 2.0685 [alpha=1e-3]",
        }
    )

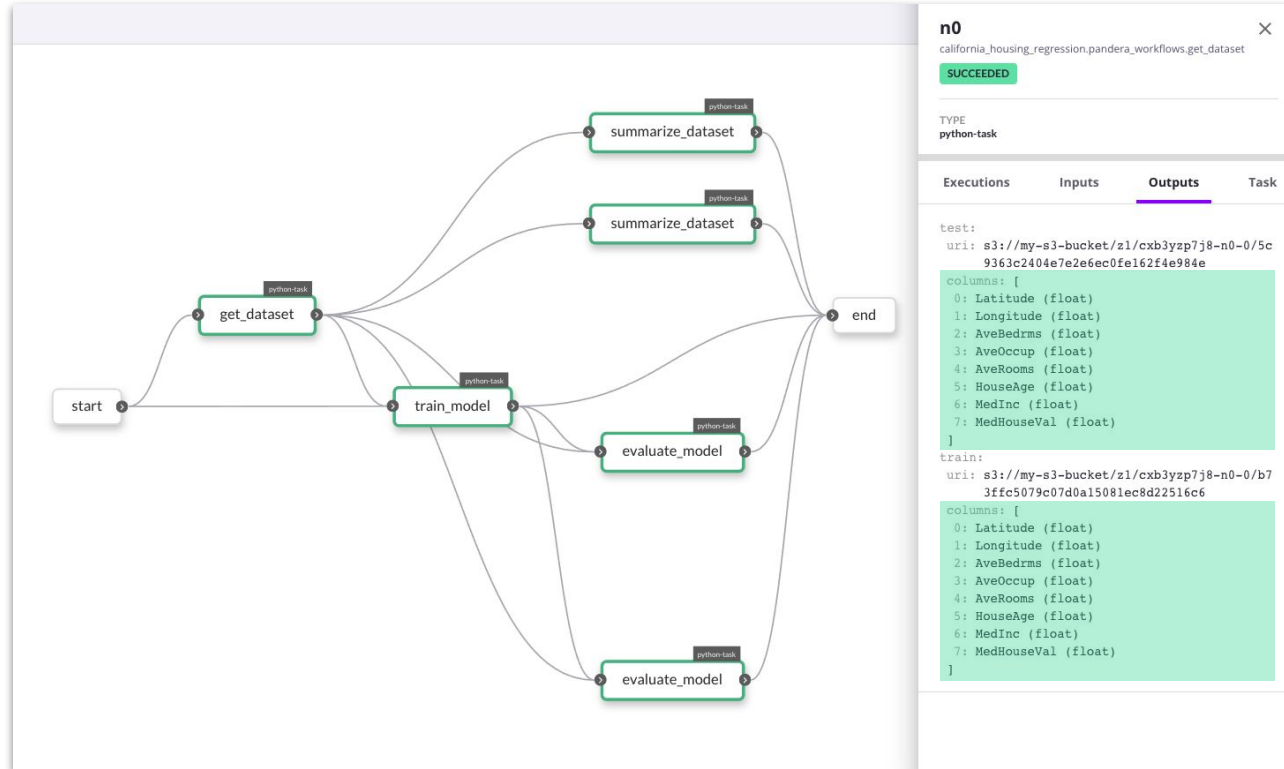
You, 4 hours ago | 1 author (You)
class Config:
    coerce = True
```

In [4]: CaliforniaHousingData.example(size=10)

Out[4]:

	Latitude	Longitude	AveBedrms	AveOccup	AveRooms	HouseAge	MedInc	MedHouseVal
0	79.860317	-148.351836	617031.632900	430128.971850	443742.477199	406612.645892	131667.872588	2.086141
1	-83.911119	-152.615588	787734.901377	913864.745456	829602.617830	588931.710189	625756.218540	2.072502
2	-80.177753	45.788823	587724.138773	200528.372483	394840.357825	521478.101969	426340.773699	2.078287
3	-14.809506	-38.640055	243580.951957	684329.660363	606582.209433	999449.778528	630028.219752	2.090909
4	-27.059190	-151.681259	864490.359815	389024.206781	916451.018379	909982.137180	931783.406294	2.087176
5	-10.783020	-43.581889	215860.187036	894330.091919	8619.707035	454911.557053	334877.131920	2.058727
6	64.063853	110.388789	467241.509949	893325.190377	915692.697615	648908.833563	413997.494038	2.078001
7	-65.360114	-148.623687	516877.114701	832633.647027	223950.545425	617144.879712	712547.371572	2.066986
8	-43.119623	61.017426	311425.228971	86337.978370	213803.011351	282039.522190	884250.395130	2.067468
9	-74.311844	-86.239245	185285.958695	385889.718367	904564.855290	111351.354414	336936.792431	2.072606

Test Your Data...



... the Functions
That Produce
Them...

```
def test_dataset():
    kwargs = {"test_size": 0.2, "random_state": 100}
    pandera_workflows.get_dataset(**kwargs)

    for get_dataset_fn, error_regex in [
        (
            pandera_column_error_workflows.get_dataset,
            r"column 'Latitude' not in dataframe",
        ),
        (
            pandera_dtype_error_workflows.get_dataset,
            r"Could not coerce <class 'pandas.core.series.Series'> data_container into type float64",
        ),
        (
            pandera_value_error_workflows.get_dataset,
            r"failed element-wise validator 0:\s<Check in_range: in_range\(-90, 90\)>",
        ),
        (
            pandera_stats_error_workflows.get_dataset,
            r"MedHouseVal mean value is not equal to 2.0685 \[alpha=1e-3\]",
        )
    ]:
        with pytest.raises(TypeError, match=error_regex):
            get_dataset_fn(**kwargs)
```

... and the
Artifacts They
Help Create.

```
@settings(max_examples=10)
@given(pandera_workflows.CaliforniaHousingData.strategy(size=30))
def test_train_model(dataset):
    model = pandera_workflows.train_model(
        dataset=dataset,
        hyperparameters=pandera_workflows.Hyperparameters(alpha=0.1, random_state=100)
    )

    features = dataset.drop(pandera_workflows.TARGET, axis="columns")
    predictions = model.predict(features)
    assert all(isinstance(x, float) for x in predictions)
```

Takeaway 1

Flyte is an *orchestration and distributed execution* platform where **type-safety** is deeply integrated with other features, which together provide strong *reliability, efficiency, and auditability* guarantees.

Takeaway 2

With **Pandera**, you can ensure the *quality of data* flowing through your machine learning pipelines *and the correctness of those pipelines themselves* by expressing **statistical types** *directly in your codebase*.

Takeaway 3

With **Flyte** and **Pandera** combined, you can **build, deploy, and scale** these ML pipelines while enjoying the guarantee that, when things go wrong, you'll know where exactly the error occurred to help you fix it.

Flyte Roadmap

Flyte Decks: A Customizable Reporting API for your Pipeline Artifacts

ML-awareness: Intra-task model checkpointing, data labeling.

Serving Integrations: support for model serving, low latency batch workflows, model monitoring.

Pandera Roadmap

Extensibility: support for *xarray*, *jsonschema*, *pyarrow*, and more!

User Experience: more built-in checks, statistical hypothesis checks

Interoperability: tighter integrations with the python ecosystem, e.g. *fastapi*, *pydantic*, *pytest*

Where do I learn more?

Flyte

website: www.flyte.org

docs: docs.flyte.org

repo: github.com/flyteorg/flyte

Pandera

docs: pandera.readthedocs.io

repo: github.com/pandera-dev/pandera

Contact

email: niels@union.ai

twitter: [@cosmicbboy](https://twitter.com/cosmicbboy)

linkedin: linkedin.com/in/nbantilan