Imports

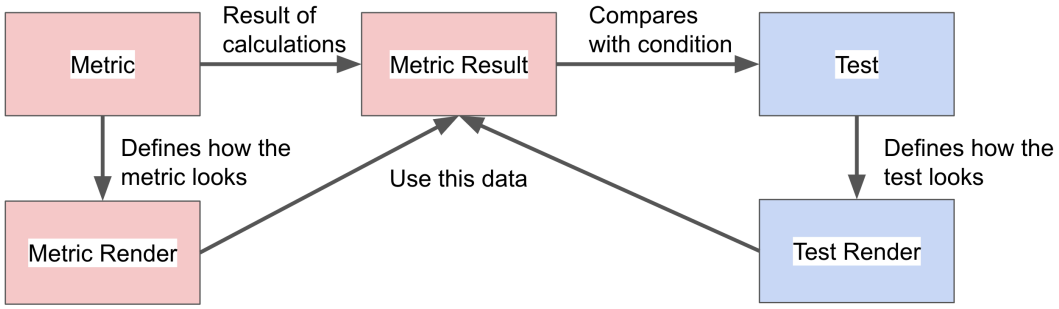
Import the required components:

In [1]:
```
from typing import List, Optional, Union
import dataclasses
import pandas as pd

from evidently.base_metric import InputData
from evidently.base_metric import Metric
from evidently.base_metric import MetricResult
from evidently.model.widget import BaseWidgetInfo
from evidently.renderers.base_renderer import MetricRenderer
from evidently.renderers.base_renderer import default_renderer
from evidently.renderers.html_widgets import CounterData
from evidently.renderers.html_widgets import header_text
from evidently.renderers.html_widgets import plotly_figure
```

Understand the architecture

The `metric` is a key component of Evidently. Each `test` uses a metric for calculations. If you want to create a test, you need to create a metric first. Both tests and metrics have `renders` which might look differently. If you are creating metric or test for your internal use, you might skip some steps: e.g., do not create a sophisticated visualization if you do not need it.



Create a new metric

Let's imagine you want to create a metric that sums up all the values in a column.

First, you need to define the resulting `dataclass`.

In [8]:
```
class MyMetricResult(MetricResult):
    class Config:
        type_alias = "evidently:metric_result:MyMetricResult"
    sum_value: float
```

Then, you need to create the class of the `metric` itself. It should have the `calculate` method which takes `InputData` - a class that contains reference dataset, current dataset and column mapping.

In [11]:
```
class MyMetric(Metric[MyMetricResult]):
  class Config:
    type_alias = "evidently:metric:MyMetric"
  column_name: str

  def __init__(self, column_name: str):
    self.column_name = column_name
    super().__init__()

  def calculate(self, data: InputData) -> MyMetricResult:
    metric_value = data.current_data[self.column_name].sum()
    return MyMetricResult(
        sum_value = metric_value
    )
```

Define the metric render

Next, you need to define the way the metric will look in the HTML reports or JSON export. Let's make a `render` class!

Note: HTML render is optional. You can skip it if you do not plan to use the metric independently, and only want to call it as part of the test.

In [15]:
```
@default_renderer(wrap_type=MyMetric)
class MyMetricRenderer(MetricRenderer):
    def render_json(self, obj: MyMetric) -> dict:
        result = dataclasses.asdict(obj.get_result())
        return result

    def render_html(self, obj: MyMetric) -> List[BaseWidgetInfo]:
        metric_result = obj.get_result()
        return [
            # helper function for visualisation. More options here More o
            header_text(label=f"My metrics value is {metric_result.sum_va
        ]
```

In [17]:
```
adult = pd.read_csv("adult.csv", header=0)
adult.head()
```

Out[17]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relation
0	25.0	Private	226802.0	11th	7.0	Never-married	Machine-op-inspct	Own-
1	38.0	Private	89814.0	HS-grad	9.0	Married-civ-spouse	Farming-fishing	Hus
2	28.0	Local-gov	336951.0	Assoc-acdm	12.0	Married-civ-spouse	Protective-serv	Hus
3	44.0	Private	160323.0	Some-college	10.0	Married-civ-spouse	Machine-op-inspct	Hus
4	18.0	NaN	103497.0	Some-college	10.0	Never-married	NaN	Own-

Here is how the metric output looks if you apply it on a sample data:

In [20]:
```
from sklearn import datasets
from evidently import ColumnMapping
from evidently.report import Report

# adult_data = datasets.fetch_openml(name='adult', version=2, as_frame='a
# adult = adult_data.frame

data_drift_dataset_report = Report(metrics=[
    MyMetric(column_name='age')
])

data_drift_dataset_report.run(reference_data=None, current_data=adult)
data_drift_dataset_report
```

Out[20]:

Loading...

In the previous step, we used a basic visualization. You might want to add more information to the widget: for example, pass the column name, show the metric value for the reference dataset, or add some visualizations (using Plotly).

To do that, you need to get these values in the `metric` class and pass to the `render` using `MetricResult`.

In [22]:
```
from plotly import graph_objs as go


class MyMetricResult(MetricResult):
```

```
    class Config:
        type_alias = "evidently:metric_result:MyMetricResult"
    feature_name: str
    current_sum_value: float
    x_values_for_hist: list
    y_values_for_hist: list
    reference_sum_value: Optional[float] # reference data could absence s


class MyMetric(Metric[MyMetricResult]):
  class Config:
    type_alias = "evidently:metric:MyMetric"
  column_name: str

  def __init__(self, column_name: str) -> None:
    self.column_name = column_name
    super().__init__()

  def calculate(self, data: InputData) -> MyMetricResult:
    reference_sum_value = None
    if data.reference_data is not None:
      reference_sum_value = data.reference_data[self.column_name].sum()
    current_sum_value = data.current_data[self.column_name].sum()
    # let's pretend we calculate some data for plot
    x_values_for_hist = [1, 2]
    y_values_for_hist = [2, 4]
    return MyMetricResult(
        feature_name = self.column_name,
        current_sum_value = current_sum_value,
        x_values_for_hist = x_values_for_hist,
        y_values_for_hist = y_values_for_hist,
        reference_sum_value = reference_sum_value
    )


@default_renderer(wrap_type=MyMetric)
class MyMetricRenderer(MetricRenderer):
    def render_json(self, obj: MyMetric, include_render: bool = False,
        include: "IncludeOptions" = None, exclude: "IncludeOptions" = Non
        result = obj.get_result().get_dict(include_render, include, exclu
        # we don't need plot data here
        result.pop("x_values_for_hist", None)
        result.pop("y_values_for_hist", None)
        return result

    def render_html(self, obj: MyMetric) -> List[BaseWidgetInfo]:
        metric_result = obj.get_result()
        figure = go.Figure(go.Bar(x=metric_result.x_values_for_hist, y=me

        return [
            header_text(label=f"The sum of '{metric_result.feature_name}'
            header_text(label=f"The sum of '{metric_result.feature_name}'
            plotly_figure(title="Example plot", figure=figure)
        ]
```

```
/opt/anaconda3/lib/python3.12/site-packages/evidently/pydantic_utils.py:13
3: UserWarning: Duplicate key (<class 'evidently.pydantic_utils.Polymorphi
cModel'>, 'evidently:metric_result:MyMetricResult') in alias map
  warnings.warn(f"Duplicate key {key} in alias map")
/opt/anaconda3/lib/python3.12/site-packages/evidently/pydantic_utils.py:13
3: UserWarning: Duplicate key (<class 'evidently.pydantic_utils.Polymorphi
cModel'>, 'evidently:metric:MyMetric') in alias map
  warnings.warn(f"Duplicate key {key} in alias map")
```

Use the metric

Here is how you can include the new metric in the Report.

In [27]:
```
data_drift_dataset_report = Report(metrics=[
    MyMetric(column_name='age')
])

data_drift_dataset_report.run(reference_data=None, current_data=adult)
data_drift_dataset_report
```

Out[27]:

Loading...

In [29]:
```
data_drift_dataset_report.json()
```

Out[29]:
```
'{"version": "0.4.30", "metrics": [{"metric": "MyMetric", "result": {"fe
ature_name": "age", "current_sum_value": 1887430.0, "reference_sum_valu
e": null}}], "timestamp": "2025-01-16 17:55:30.113021"}'
```

Create a new test

If you want to be able to compare the new metric against a defined condition as part of a Test Suite, you need to create a Test.

To make a Test, you need a Metric:

- The metric calculates the value
- The test gets the values, and performs the comparison

We already got the metric. Now, let's make a test.

When you create a test, you can also define the default test conditions. They will apply if you call the test "as is" without passing a custom constraint.

Note: for simple simple scalar functions instead of `MyMetric` and `MyTest` you can use `CustomValueMetric` and `CustomValueTest` respectively.

```
CustomValueTest(func=<function(InputData)->float>, title='custom
test', lte=0.2)
```

In [33]:

```python
from abc import ABC
from evidently.utils.types import Numeric
from evidently.renderers.base_renderer import TestHtmlInfo
from evidently.renderers.base_renderer import TestRenderer
from evidently.tests.base_test import BaseCheckValueTest
from evidently.tests.base_test import GroupData
from evidently.tests.base_test import GroupingTypes
from evidently.tests.base_test import TestValueCondition

# make a group for test. It used for grouping tests in the report
MY_GROUP = GroupData("my_group", "My Group", "")
GroupingTypes.TestGroup.add_value(MY_GROUP)

class MyTest(BaseCheckValueTest, ABC):
    name = "My test"
    group = MY_GROUP.id

    column_name: str
    # define a metric used for calculation
    _metric: MyMetric

    def __init__(
        self,
        column_name: str,
        eq: Optional[Numeric] = None,
        gt: Optional[Numeric] = None,
        gte: Optional[Numeric] = None,
        is_in: Optional[List[Union[Numeric, str, bool]]] = None,
        lt: Optional[Numeric] = None,
        lte: Optional[Numeric] = None,
        not_eq: Optional[Numeric] = None,
        not_in: Optional[List[Union[Numeric, str, bool]]] = None,
    ):
        self.column_name = column_name
        super().__init__(eq=eq, gt=gt, gte=gte, is_in=is_in, lt=lt, lte=l
        self._metric = MyMetric(self.column_name)

    def get_condition(self) -> TestValueCondition:
        # if condition specified like lte=8 or gt=3 etc
        if self.condition.has_condition():
            return self.condition
        # if there is no condition but we have reference and we want to c
        ref_result = self._metric.get_result().reference_sum_value
        if ref_result is not None:
          return TestValueCondition(lte=ref_result)
        # if there is no condition, no reference data but we have some id
        return TestValueCondition(gt=0)

    # define the value we will compare against condition
    def calculate_value_for_test(self) -> Numeric:
        return self._metric.get_result().current_sum_value
    # define the way test will look like in a table
    def get_description(self, value: Numeric) -> str:
        return f"The sum of '{self._metric.get_result().feature_name}' co
```

You can add a `render` class for the test as well. This class also should use data from the metric result only.

Note: it's optional. You can still use the test if you do not define the render. In this case, Evidently will use the information from the 'get_description' method to show the test output in the preview. However, once you click on "details" there will be no supporting visualization.

In [36]:
```
@default_renderer(wrap_type=MyTest)
class MyTestRenderer(TestRenderer):
    def render_json(self, obj: MyTest) -> dict:
        result = super().render_json(obj)
        metric_result = obj._metric.get_result()
        result["parameters"]["condition"] = obj.get_condition().as_dict()
        result["parameters"]["reference_sum_value"] = metric_result.refer
        result["parameters"]["current_sum_value"] = metric_result.current
        return result

    def render_html(self, obj: MyTest) -> List[BaseWidgetInfo]:
        info = super().render_html(obj)
        metric_result = obj._metric.get_result()
        figure = go.Figure(go.Bar(x=metric_result.x_values_for_hist, y=me
        info.with_details("", plotly_figure(title="Example plot", figure=
        return info
```

Use the new test

Here is how you can include your new test in a Test Suite:

In [40]:
```
from evidently.test_suite import TestSuite

my_tests = TestSuite(tests=[
    MyTest(column_name='age')
])

my_tests.run(reference_data=adult[:5000], current_data=adult[5000:10000])
my_tests
```

Out[40]:

Loading...

Create a custom report with the new metric

You can combine your new metric with other metrics available in the library in a single report.

In [44]:
```
from evidently.metrics import *

data_drift_dataset_report = Report(metrics=[
    MyMetric(column_name='age'),
    ColumnDriftMetric(column_name='age'),
    DatasetMissingValuesMetric(),

])

data_drift_dataset_report.run(reference_data=adult[:5000], current_data=a
data_drift_dataset_report
```

Out[44]:

Loading...

Create a custom test suite with a new test

It works the same way for tests.

In [48]:
```
from evidently.tests import *


my_tests = TestSuite(tests=[
    MyTest(column_name='age'),
    TestNumberOfRowsWithMissingValues(),
    TestNumberOfConstantColumns()

])

my_tests.run(reference_data=adult[:5000], current_data=adult[5000:10000])
my_tests
```

Out[48]:

Loading...

In [ ]:
```

```

In [51]:
```
import dataclasses
import pandas as pd

import re
import numpy as np
```

```
from math import ceil

from abc import ABC
from evidently.utils.types import Numeric
from evidently.renderers.base_renderer import TestHtmlInfo, TestRenderer
from evidently.tests.base_test import BaseCheckValueTest, GroupData, Grou


from plotly import graph_objs as go
from sklearn import datasets
from typing import List, Optional, Union

from evidently.test_suite import TestSuite
from evidently.metric_preset import DataQualityPreset, DataDriftPreset
from evidently.test_preset import DataDriftTestPreset
from evidently.report import Report

from evidently.renderers.base_renderer import MetricRenderer, default_ren
from evidently.renderers.html_widgets import BaseWidgetInfo, header_text,
from plotly import graph_objs as go


from evidently import ColumnMapping
from evidently.report import Report
from evidently.base_metric import InputData
from evidently.base_metric import Metric
from evidently.base_metric import MetricResult
from evidently.model.widget import BaseWidgetInfo
from evidently.renderers.base_renderer import MetricRenderer
from evidently.renderers.base_renderer import default_renderer
from evidently.renderers.html_widgets import CounterData
from evidently.renderers.html_widgets import header_text
from evidently.renderers.html_widgets import plotly_figure
from evidently.renderers.html_widgets import WidgetSize
```

In [ ]:

In [54]:
```
import re
import numpy as np
from math import ceil
```

In [56]:
```
from evidently.ui.workspace import Workspace
```

In [62]:
```
# workspace_name = "ss_workspace_proxy_resmed_demo1"
workspace_name = "ss_workspace_proxy_resmed_demo9"
pjct_name = "smart_coachin_project_demo3"
pjct_description = "Data Drift Monitoring for Adult Income Dataset"
```

In [64]:
```
ws = Workspace(workspace_name)
projects = ws.list_projects()
projects
```

Out[64]: []

In [66]:
```
Workspace.create(workspace_name)
```

Out[66]: <evidently.ui.workspace.view.LocalWorkspaceView at 0x1799d2c00>

In [68]:
```
new_project = ws.create_project(pjct_name)
new_project.description = pjct_description
```

In [70]:
```
# ws.add_test_suite(new_project.id, my_tests)
# ws.add_test_suite(new_project.id, test_suite)
```

In [72]:
```
ws.add_report(new_project.id, data_drift_dataset_report)
ws.add_test_suite(new_project.id, my_tests)
```

In [ ]:
```

```

In [ ]:
```
!evidently ui --workspace ss_workspace_proxy_resmed_demo9/ --host 0.0.0.0
```

```
/opt/anaconda3/lib/python3.12/site-packages/evidently/ui/api/projects.py:2
55: LitestarWarning: Use of a synchronous callable <function additional_mo
dels at 0x16c0f11c0> without setting sync_to_thread is discouraged since s
ynchronous callables can block the main thread if they perform blocking op
erations. If the callable is guaranteed to be non-blocking, you can set sy
nc_to_thread=False to skip this warning, or set the environmentvariable LI
TESTAR_WARN_IMPLICIT_SYNC_TO_THREAD=0 to disable warnings of this type ent
irely.
  @get("/models/additional")
/opt/anaconda3/lib/python3.12/site-packages/evidently/ui/components/securi
ty.py:82: LitestarWarning: Use of a synchronous callable <bound method NoS
ecurityComponent.get_security of NoSecurityComponent(type='none', dummy_us
er_id=UUID('00000000-0000-0000-0000-000000000001'), dummy_org_id=UUID('000
00000-0000-0000-0000-000000000002'))> without setting sync_to_thread is di
scouraged since synchronous callables can block the main thread if they pe
rform blocking operations. If the callable is guaranteed to be non-blockin
g, you can set sync_to_thread=False to skip this warning, or set the envir
onmentvariable LITESTAR_WARN_IMPLICIT_SYNC_TO_THREAD=0 to disable warnings
of this type entirely.
  "security": Provide(self.get_security),
/opt/anaconda3/lib/python3.12/site-packages/evidently/ui/components/securi
ty.py:83: LitestarWarning: Use of a synchronous callable <function SimpleS
ecurity.get_dependencies.<locals>.<lambda> at 0x16c135a80> without setting
sync_to_thread is discouraged since synchronous callables can block the ma
in thread if they perform blocking operations. If the callable is guarante
ed to be non-blocking, you can set sync_to_thread=False to skip this warni
ng, or set the environmentvariable LITESTAR_WARN_IMPLICIT_SYNC_TO_THREAD=0
to disable warnings of this type entirely.
  "security_config": Provide(lambda: self),
/opt/anaconda3/lib/python3.12/site-packages/evidently/ui/components/securi
ty.py:84: LitestarWarning: Use of a synchronous callable <function SimpleS
ecurity.get_dependencies.<locals>.<lambda> at 0x16c159bc0> without setting
sync_to_thread is discouraged since synchronous callables can block the ma
in thread if they perform blocking operations. If the callable is guarante
ed to be non-blocking, you can set sync_to_thread=False to skip this warni
ng, or set the environmentvariable LITESTAR_WARN_IMPLICIT_SYNC_TO_THREAD=0
to disable warnings of this type entirely.
  "auth_manager": Provide(lambda: NoopAuthManager()),
INFO:     Started server process [25606]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
fatal: not a git repository (or any of the parent directories): .git
INFO:     127.0.0.1:54723 - "GET /api/version HTTP/1.1" 200 OK
INFO:     127.0.0.1:54724 - "GET /api/projects HTTP/1.1" 500 Internal Serv
er Error
```

In [ ]: