Working with MINTe: Python emulator for malaria intervention scenarios

This notebook walks through how to:

1. Install the MINTverse Python packages
2. Map common **R** idioms (MINTer / MINTweb) to **Python**
3. Run **single** and **multiple** scenarios with `run_minter_scenarios`
4. Understand the **outputs** (prevalence, cases, scenario metadata)
5. Explore results in tabular form (`.head()`, filtering, grouping)
6. Use the built-in plotting helper `create_scenario_plots`
7. Export results to `.csv` for further analysis

We treat the ML models as a **black box surrogate** for `malariasimulation`:

- You provide: baseline setting + intervention package(s)
- MINTe returns: predicted prevalence and clinical cases over time

1. Installation

You only need to run the installation **once per environment**.

In most setups you will either:

- Install from PyPI or
- Install directly from the GitHub repositories.

If you're on an HPC or managed environment, please speak to me after (Docs are not written yet for this).

```
# Let's install minte
!pip install minte

# Otherwise, install directly from GitHub:
# !pip install "git+https://github.com/CosmoNaught/MINTe-python.git"
# !pip install "git+https://github.com/CosmoNaught/estiMINT-python.git"
```

```
Collecting minte
  Downloading minte-1.0.3-py3-none-any.whl.metadata (9.6 kB)
Collecting estimint>=1.0.0 (from minte)
  Downloading estimint-1.0.0-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: joblib>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from minte) (1.5.2)
Requirement already satisfied: matplotlib>=3.7.0 in /usr/local/lib/python3.12/dist-packages (from minte) (3.10.0)
Requirement already satisfied: numpy>=1.24.0 in /usr/local/lib/python3.12/dist-packages (from minte) (2.0.2)
Requirement already satisfied: pandas>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from minte) (2.2.2)
Requirement already satisfied: scikit-learn==1.6.1 in /usr/local/lib/python3.12/dist-packages (from minte) (1.6.1)
Requirement already satisfied: scipy>=1.10.0 in /usr/local/lib/python3.12/dist-packages (from minte) (1.16.3)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from minte) (2.9.0+cu126)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn=
Requirement already satisfied: duckdb>=0.8.0 in /usr/local/lib/python3.12/dist-packages (from estimint>=1.0.0->min
Requirement already satisfied: xgboost>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from estimint>=1.0.0->mi
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.7.0
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.7.0->mi
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.7.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.7.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.7.0-
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.7.0->minte
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3.7.0
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib>=3
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.0.0->minte)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.0.0->mint
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->minte) (3.2
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from torch>=2
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->minte) (7
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->minte)
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->mint
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->minte) (3.1.6
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->minte)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from to
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from to
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch>
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from tor
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from tor
```

```
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch>=2.
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch>
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch>=2
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from tor
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->minte)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->mat
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>=2.0
Downloading minte-1.0.3-py3-none-any.whl (45.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.5/45.5 MB 15.0 MB/s eta 0:00:00
Downloading estimint-1.0.0-py3-none-any.whl (4.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.7/4.7 MB 58.4 MB/s eta 0:00:00
Installing collected packages: estimint, minte
Successfully installed estimint-1.0.0 minte-1.0.3
```

2. Common R → Python equivalents

Most of what you did in R with **MINTer / MINTweb** and **tidyverse** has a direct analogue in **Python + pandas**.

Task	R / tidyverse	Python / pandas
Data frame	`data.frame()`, `tibble()`	`pd.DataFrame()`
Read CSV	`readr::read_csv("x.csv")`	`pd.read_csv("x.csv")`
Filter rows	`df %>% filter(var == 1)`	`df[df["var"] == 1]`
Select columns	`df %>% select(a, b)`	`df[["a", "b"]]`
Arrange / sort	`df %>% arrange(a)`	`df.sort_values("a")`
Grouped summary	`df %>% group_by(a) %>% summarise(mean(b))`	`df.groupby("a")["b"].mean().reset_index()`
Pipe	`%>%`	chain methods: `df.query(...).groupby(...).mean()`
Run scenarios	`run_mintweb_controller(...)`	`run_minter_scenarios(...)`
Missing value (numeric)	`NA_real_`	`numpy.nan` (imported as `np.nan`)
Missing value (string)	`NA_character_`	`None`

Conceptually:

- In R you passed **vectors** (`c(0.3, 0.5, 0.7)`);
  in Python you pass **lists** or **NumPy arrays** (`[0.3, 0.5, 0.7]` or `np.array([...])`).
- `run_minter_scenarios` is the Python analogue of `run_mintweb_controller`: you give it vectors of parameters, and it runs **all scenarios at once**.

3. Imports and basic configuration

Here we import:

- `numpy` and `pandas` for data handling
- `run_minter_scenarios` – the main controller
- `create_scenario_plots` – a built-in plotting helper

```
import numpy as np
import pandas as pd

# Core MINTe API
from minte import run_minter_scenarios, create_scenario_plots

# Optional: make pandas print a bit more information
pd.set_option("display.max_columns", 50)
pd.set_option("display.width", 120)
```

4. What `run_minter_scenarios` does

At a high level, `run_minter_scenarios`:

1. **Back-calculates EIR** from current prevalence and interventions
   using the pre-trained **estiMINT XGBoost** model.
2. Builds a **scenario table** with:
   - baseline EIR
   - current and future ITN/IRS/LSM

   - vector behaviour (`Q0`, `phi_bednets`)
   - resistance & net quality (`dn0_use`, `dn0_future`)
3. Runs the **neural emulator** to predict:
   - under-5 daily prevalence trajectories
   - all-age daily clinical incidence trajectories per 1000
4. Returns a results object (similar to an R list) with:

```
results.prevalence   # DataFrame of prevalence over time for each scenario
results.cases        # DataFrame of clinical cases over time for each scenario
results.scenario_meta# Per-scenario metadata, incl. EIR validity
results.eir_valid    # True/False flag
results.benchmarks   # (optional) runtime timings
```

In this notebook we treat the ML models as a **black box**: you don't have to write or edit any neural-network code to use MINTe.

5. A single simple scenario

Here we run **one** scenario by passing **1-element lists**. This is conceptually the same as a one-row data frame in R.

```
# Example: a single scenario
scenario_tag   = ["example_scenario"]
res_use        = [0.2]   # current resistance
py_only        = [0.3]
py_pbo         = [0.2]
py_pyrrole     = [0.1]
py_ppf         = [0.05]

prev           = [0.55]  # current under-5 prevalence at decision time
Q0             = [0.92]  # proportion of bites indoors
phi            = [0.85]  # proportion of bites while people are in bed
season         = [0]     # 0 = perennial, 1 = strongly seasonal

irs            = [0.4]   # current IRS coverage
irs_future     = [0.4]   # future IRS coverage
lsm            = [0.2]   # future LSM coverage
routine        = [1]     # 1 = routine ITN distribution on, 0 = off

# Future ITNs: here we scale up py-only nets to 45% coverage
itn_future      = [0.45]
net_type_future = ["py_only"]

res_one = run_minter_scenarios(
    scenario_tag=scenario_tag,
    res_use=res_use,
    py_only=py_only,
    py_pbo=py_pbo,
    py_pyrrole=py_pyrrole,
    py_ppf=py_ppf,
    prev=prev,
    Q0=Q0,
    phi=phi,
    season=season,
    irs=irs,
    itn_future=itn_future,
    net_type_future=net_type_future,
    irs_future=irs_future,
    routine=routine,
    lsm=lsm,
)

res_one
```

```
=== Benchmark Results ===
Pre-load models to cache: 0.144 seconds
Run EIR predictions (1 scenarios): 0.248 seconds
Run Prevalence NN (1 scenarios): 0.217 seconds
Run Cases NN (1 scenarios): 0.084 seconds

Total time: 0.696 seconds
==============================
```

```
MinterResults(prevalence=     index  timestep  prevalence model_type          scenario      scenario_tag
eir_valid
0        0         1    0.545952       LSTM  example_scenario  example_scenario       True
1        0         2    0.555139       LSTM  example_scenario  example_scenario       True
2        0         3    0.546531       LSTM  example_scenario  example_scenario       True
3        0         4    0.532927       LSTM  example_scenario  example_scenario       True
4        0         5    0.514411       LSTM  example_scenario  example_scenario       True
..     ...       ...         ...        ...               ...               ...        ...
151      0       152    0.578962       LSTM  example_scenario  example_scenario       True
152      0       153    0.582537       LSTM  example_scenario  example_scenario       True
153      0       154    0.586865       LSTM  example_scenario  example_scenario       True
154      0       155    0.594491       LSTM  example_scenario  example_scenario       True
155      0       156    0.604944       LSTM  example_scenario  example_scenario       True

[156 rows x 7 columns], cases=     index  timestep     cases model_type          scenario      scenario_tag
eir_valid
0        0         1  2.511641       LSTM  example_scenario  example_scenario       True
1        0         2  1.574790       LSTM  example_scenario  example_scenario       True
2        0         3  0.791433       LSTM  example_scenario  example_scenario       True
3        0         4  0.534768       LSTM  example_scenario  example_scenario       True
4        0         5  0.438171       LSTM  example_scenario  example_scenario       True
..     ...       ...       ...        ...               ...               ...        ...
151      0       152  2.534444       LSTM  example_scenario  example_scenario       True
152      0       153  2.412336       LSTM  example_scenario  example_scenario       True
153      0       154  2.421351       LSTM  example_scenario  example_scenario       True
154      0       155  2.667779       LSTM  example_scenario  example_scenario       True
155      0       156  3.157067       LSTM  example_scenario  example_scenario       True

[156 rows x 7 columns], scenario_meta=       scenario_tag  eir_valid
0  example_scenario       True, eir_valid=True, benchmarks={'preload_models': 0.14382243156433105,
'run_eir_models': 0.24842548370361328, 'run_neural_network_prevalence': 0.21695184707641602,
'run_neural_network_cases': 0.08445024490356445, 'total': 0.6956417560577393, 'total_scenarios': 1})
```

6. What MINTe returns (`res.prevalence` and `res.cases`)

The result object exposes the main outputs as attributes:

- `res.prevalence` – DataFrame with columns like:
  - `index` (scenario index)
  - `timestep` (time index, in 14-day steps)
  - `prevalence` (under-5 prevalence)
  - `model_type` (e.g. `"LSTM"`)
  - `scenario` / `scenario_tag`
  - `eir_valid` (whether the EIR is inside the calibrated range)
- `res.cases` – DataFrame with similar structure, but with `cases` instead of `prevalence`

Let's look at the first few rows to get a feel for this structure.

```
print("Prevalence (head):")
display(res_one.prevalence.head())

print("\nCases (head):")
display(res_one.cases.head())

print("\nColumns in prevalence table:", list(res_one.prevalence.columns))
print("Columns in cases table:", list(res_one.cases.columns))
```

```
Prevalence (head):
```

	index	timestep	prevalence	model_type	scenario	scenario_tag	eir_valid
0	0	1	0.545952	LSTM	example_scenario	example_scenario	True
1	0	2	0.555139	LSTM	example_scenario	example_scenario	True
2	0	3	0.546531	LSTM	example_scenario	example_scenario	True
3	0	4	0.532927	LSTM	example_scenario	example_scenario	True
4	0	5	0.514411	LSTM	example_scenario	example_scenario	True

```
Cases (head):
```

	index	timestep	cases	model_type	scenario	scenario_tag	eir_valid
0	0	1	2.511641	LSTM	example_scenario	example_scenario	True
1	0	2	1.574790	LSTM	example_scenario	example_scenario	True
2	0	3	0.791433	LSTM	example_scenario	example_scenario	True
3	0	4	0.534768	LSTM	example_scenario	example_scenario	True
4	0	5	0.438171	LSTM	example_scenario	example_scenario	True

```
Columns in prevalence table: ['index', 'timestep', 'prevalence', 'model_type', 'scenario', 'scenario_tag', 'eir_va
Columns in cases table: ['index', 'timestep', 'cases', 'model_type', 'scenario', 'scenario_tag', 'eir_valid']
```

7. Running multiple scenarios (R `run_mintweb_controller` → Python)

In R you might have run something like:

```
high_prev <- run_mintweb_controller(
  scenario_tag   = c("no_intervention", "irs_only", ...),
  res_use        = c(0.2, 0.2, ...),
  ...
)
```

The Python equivalent is to pass **lists** of equal length to `run_minter_scenarios`. Each position `i` defines one scenario.

Below we reproduce the high-prevalence example in Python.

```
# High-prevalence example with multiple intervention packages
scenario_tag = [
    "no_intervention", "irs_only", "lsm_only", "py_only_only",
    "py_only_with_lsm", "py_pbo_only", "py_pbo_with_lsm",
    "py_pyrrole_only", "py_pyrrole_with_lsm", "py_ppf_only",
    "py_ppf_with_lsm",
]

n = len(scenario_tag)

res_use       = [0.2] * n
py_only       = [0.3] * n
py_pbo        = [0.2] * n
py_pyrrole    = [0.1] * n
py_ppf        = [0.05] * n

prev          = [0.55] * n
Q0            = [0.92] * n
phi           = [0.85] * n
season        = [0]    * n

irs           = [0.4]  * n

itn_future = [
    0.00, 0.00, 0.00,   # no nets for the first three scenarios
    0.45, 0.45,         # py_only w/wo LSM
    0.45, 0.45,         # py_pbo w/wo LSM
    0.45, 0.45,         # py_pyrrole w/wo LSM
    0.45, 0.45,         # py_ppf w/wo LSM
]

net_type_future = [
    None, None, None,
```

```
    "py_only", "py_only",
    "py_pbo", "py_pbo",
    "py_pyrrole", "py_pyrrole",
    "py_ppf", "py_ppf",
]

irs_future = [
    0.0, 0.5, 0.0,      # second scenario increases IRS
    0.0, 0.0,
    0.0, 0.0,
    0.0, 0.0,
    0.0, 0.0,
]

routine = [
    0, 0, 0,           # first three: no routine distribution
    1, 1,
    1, 1,
    1, 1,
    1, 1,
]

lsm = [
    0.0, 0.0, 0.2,     # third scenario: LSM only
    0.0, 0.2,
    0.0, 0.2,
    0.0, 0.2,
    0.0, 0.2,
]

res = run_minter_scenarios(
    scenario_tag    = scenario_tag,
    res_use         = res_use,
    py_only         = py_only,
    py_pbo          = py_pbo,
    py_pyrrole      = py_pyrrole,
    py_ppf          = py_ppf,
    prev            = prev,
    Q0              = Q0,
    phi             = phi,
    season          = season,
    irs             = irs,
    itn_future      = itn_future,
    net_type_future= net_type_future,
    irs_future      = irs_future,
    routine         = routine,
    lsm             = lsm,
)

print("Prevalence shape:", res.prevalence.shape)
print("Cases shape:", res.cases.shape)

res.prevalence.head()
```

```
=== Benchmark Results ===
Pre-load models to cache: 0.000 seconds
Run EIR predictions (11 scenarios): 0.192 seconds
Run Prevalence NN (11 scenarios): 0.097 seconds
Run Cases NN (11 scenarios): 0.146 seconds

Total time: 0.437 seconds
==============================
Prevalence shape: (1716, 7)
Cases shape: (1716, 7)
```

	index	timestep	prevalence	model_type	scenario	scenario_tag	eir_valid
0	0	1	0.545952	LSTM	no_intervention	no_intervention	True
1	0	2	0.555139	LSTM	no_intervention	no_intervention	True
2	0	3	0.546531	LSTM	no_intervention	no_intervention	True
3	0	4	0.532927	LSTM	no_intervention	no_intervention	True
4	0	5	0.514411	LSTM	no_intervention	no_intervention	True

8. Exploring results in data form

Typical tasks a malaria researcher might want:

- Look at the first few rows: `.head()`
- Filter to a specific scenario or time window
- Summarise average prevalence / incidence over a period
- Compare scenarios side-by-side

We do this with **pandas**, which plays the same role as **dplyr** in R.

```
prev_df = res.prevalence.copy()
cases_df = res.cases.copy()

# First few rows of each
print("Prevalence:")
display(prev_df.head())

print("\nCases:")
display(cases_df.head())

# Unique scenarios
print("\nScenarios:", prev_df["scenario"].unique())

# Example: subset to a single scenario
subset = prev_df[prev_df["scenario"] == "py_pbo_with_lsm"]
display(subset.head())

# Example: summary over the whole time horizon
mean_prev_by_scenario = (
    prev_df.groupby("scenario")["prevalence"]
    .mean()
    .reset_index()
    .sort_values("prevalence", ascending=False)
)

print("\nMean prevalence over all timesteps by scenario:")
display(mean_prev_by_scenario)
```

```
Prevalence:
```

	index	timestep	prevalence	model_type	scenario	scenario_tag	eir_valid
0	0	1	0.545952	LSTM	no_intervention	no_intervention	True
1	0	2	0.555139	LSTM	no_intervention	no_intervention	True
2	0	3	0.546531	LSTM	no_intervention	no_intervention	True
3	0	4	0.532927	LSTM	no_intervention	no_intervention	True
4	0	5	0.514411	LSTM	no_intervention	no_intervention	True

```
Cases:
```

	index	timestep	cases	model_type	scenario	scenario_tag	eir_valid
0	0	1	2.511641	LSTM	no_intervention	no_intervention	True
1	0	2	1.574790	LSTM	no_intervention	no_intervention	True
2	0	3	0.791433	LSTM	no_intervention	no_intervention	True
3	0	4	0.534768	LSTM	no_intervention	no_intervention	True
4	0	5	0.438171	LSTM	no_intervention	no_intervention	True

```
Scenarios: ['no_intervention' 'irs_only' 'lsm_only' 'py_only_only' 'py_only_with_lsm'
 'py_pbo_only' 'py_pbo_with_lsm' 'py_pyrrole_only' 'py_pyrrole_with_lsm'
 'py_ppf_only' 'py_ppf_with_lsm']
```

	index	timestep	prevalence	model_type	scenario	scenario_tag	eir_valid
936	6	1	0.545952	LSTM	py_pbo_with_lsm	py_pbo_with_lsm	True
937	6	2	0.555139	LSTM	py_pbo_with_lsm	py_pbo_with_lsm	True
938	6	3	0.546531	LSTM	py_pbo_with_lsm	py_pbo_with_lsm	True
939	6	4	0.532927	LSTM	py_pbo_with_lsm	py_pbo_with_lsm	True
940	6	5	0.514411	LSTM	py_pbo_with_lsm	py_pbo_with_lsm	True

```
Mean prevalence over all timesteps by scenario:
```

	scenario	prevalence
2	no_intervention	0.663541
3	py_only_only	0.653043
1	lsm_only	0.652264
7	py_ppf_only	0.646333
4	py_only_with_lsm	0.638673
5	py_pbo_only	0.638545
9	py_pyrrole_only	0.633692
8	py_ppf_with_lsm	0.631508
6	py_pbo_with_lsm	0.623373
10	py_pyrrole_with_lsm	0.618348
0	irs_only	0.522648

Next steps: Generate code with `mean_prev_by_scenario` New interactive sheet

9. Using the built-in plotting helper: `create_scenario_plots`

MINTe provides a convenience function `create_scenario_plots` that:

- Takes the `res.prevalence` (and/or `res.cases`) table
- Automatically generates per-scenario plots of prevalence and/or cases over time
- Saves them as image files (e.g. `.png`) in a chosen folder

This is the quickest way to get a full set of figures for a gallery of scenarios.

```
import os


# Create a folder for plots (if it doesn't exist)
os.makedirs("plots", exist_ok=True)
```

```python
plots = create_scenario_plots(
    res.prevalence,
    output_dir="plots/",
    plot_type="both",   # "prevalence", "cases", or
)

print("Created plots:", plots)
```