

# Code pipelines in Python

## Managing code with Make

2025-03-23

# Strategies for reproducibility



@allison\_horst

Art by Allison Horst

# Clean execution environments

## *Quarto documents*

In this exercise, we'll look at a report generated completely in Python using Quarto.

We're using an intentionally simplified report: what we're telling the readers, how we fit models, and so on, are all much simpler than you would use in practice. Instead, focus on the big idea here: reproducible reports!

▷ Run Cell | Run Next Cell

```
```{python}
#/ label: setup
#/ include: false
import polars as pl
import polars.selectors as cs
from great_tables import GT
import pickle
...```

```

# Clean execution environments

*Fresh Jupyter Kernels*



# Strategies for reproducibility

- Quarto for reproducible documents
- Virtual environments for package version management, such as conda or uv
- git/GitHub for version control

# Script-oriented workflows

01-read-data.py

02-clean-data.py

03-descriptive-stats.py

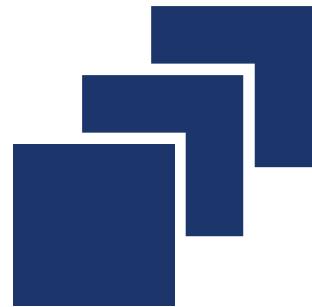
...

n-output.py

report.qmd

- Doesn't scale well—in terms of both time and scope
- Not clear what we can skip

# GNU Make



**Scale** the work  
you need.



**Skip** the work  
you don't.



**See** evidence  
of reproducibility.

# Setting up a pipeline

- Make requires a `Makefile` file in the root directory of your project

### Makefile

```
1 figures/gapminder_plot.png: scripts/create_line_plot.py data/gapminder.csv  
2     uv run scripts/create_line_plot.py
```

```
└── data  
    └── gapminder.csv  
└── scripts  
    └── create_line_plot.py  
└── Makefile
```

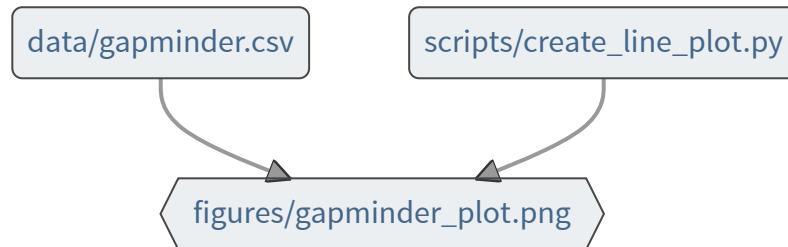
```
scripts/create_line_plot.py
```

```
1 def create_line_plot(df: pl.DataFrame) -> plt.Figure:
2     ## -- snip --
3     return fig
4
5 def main():
6     gapminder = pl.read_csv("data/gapminder.csv")
7     fig = create_line_plot(gapminder)
8
9     # Ensure the output directory exists.
10    os.makedirs("figures", exist_ok=True)
11
12    # Save the figure as a PNG image.
13    fig.savefig("figures/gapminder_plot.png", dpi=300, bbox_inches="tight")
14    plt.close(fig)
15
16 if __name__ == "__main__":
17     main()
```

# Visualizing Makefile

## Makefile

```
1 figures/gapminder_plot.png: scripts/create_line_plot.py data/gapminder.csv  
2     uv run scripts/create_line_plot.py
```



# What's Make's plan?

```
1 make --dry-run
```

```
1 uv run scripts/create_line_plot.py
```

Run this in the terminal! Don't put it in Makefile or other scripts.

# What's Make's plan?

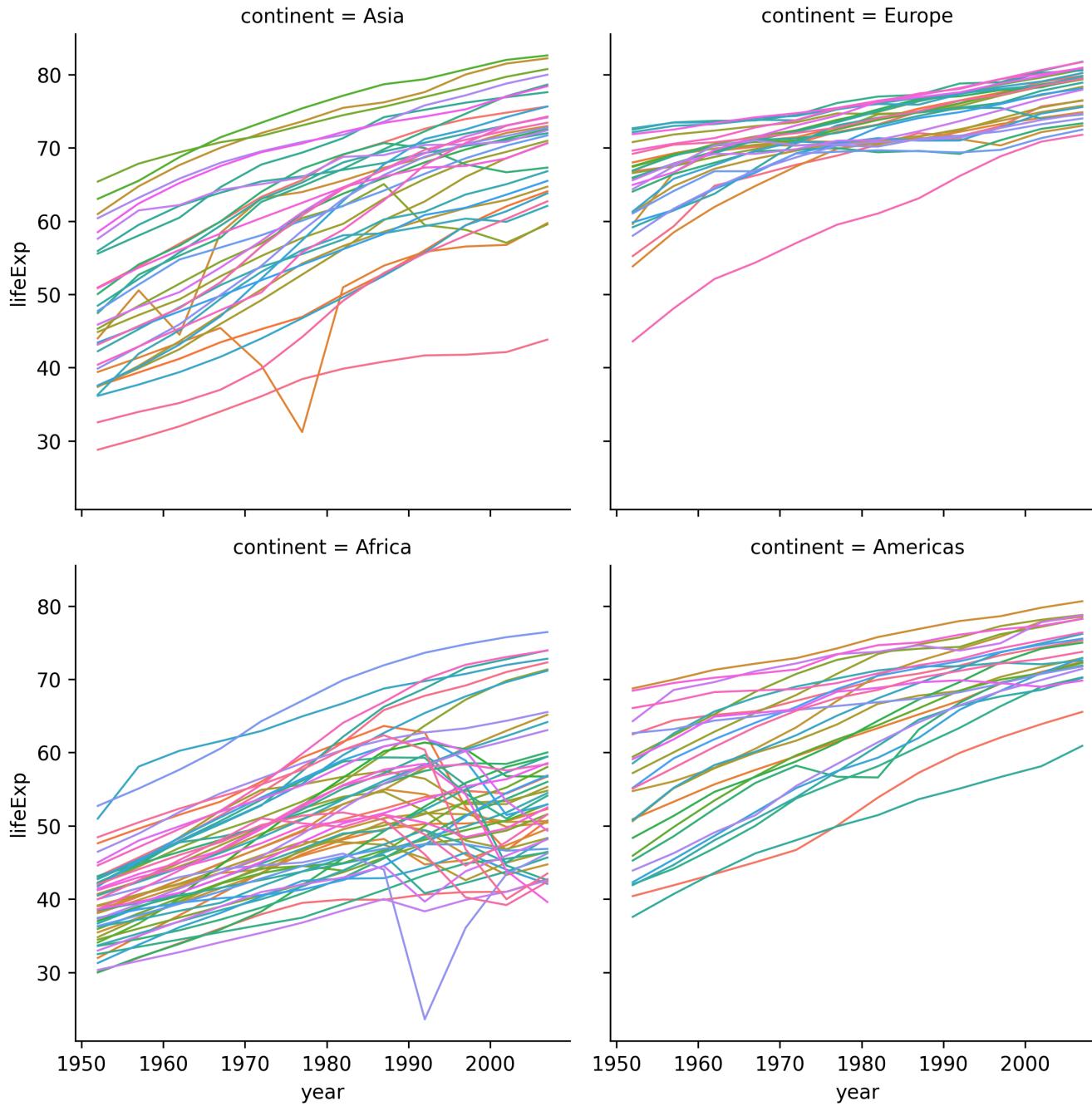
```
1 make -n
```

# Building the pipeline

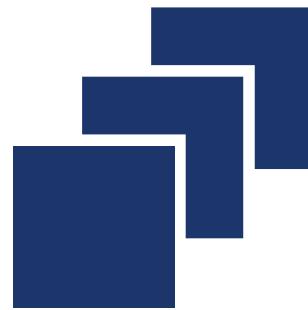
1 make

```
uv run scripts/create_line_plot.py
```

```
├── Makefile
├── data
│   └── gapminder.csv
├── figures
│   └── gapminder_plot.png
└── scripts
    └── create_line_plot.py
```



# GNU Make



**Scale** the work  
you need.



**Skip** the work  
you don't.



**See** evidence  
of reproducibility.

## ***Your Turn 1***

**Work through *Your Turn 1* in **exercises.qmd****

# Makefile

```
1 target: dependency  
2     recipe
```

The indent *must* be a tab!

# Makefile

```
1 target: dependency1 dependency2  
2     recipe
```

# Makefile

```
1 target1: target2  
2     recipel  
3  
4 target2: dependency1 dependency2  
5     recipe2
```

# Makefile

```
1 all: target2  
2  
3 target1: dependency1 dependency2  
4     recipel  
5  
6 target2: target1  
7     recipe2
```

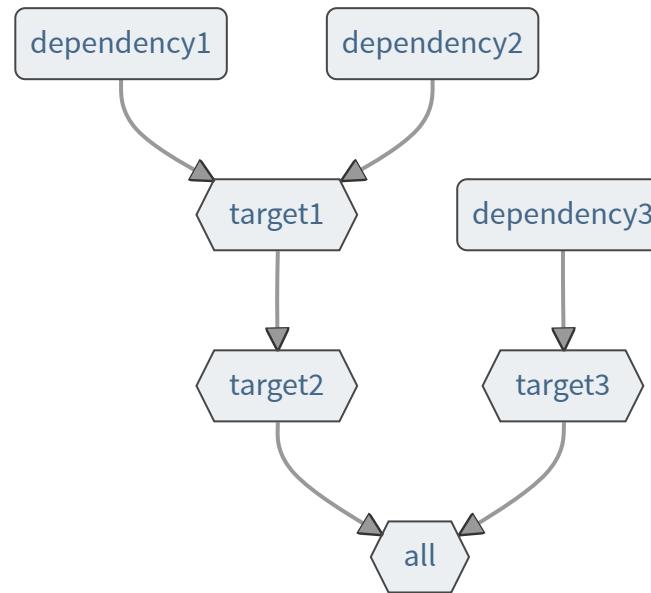
# Makefile

```
1 all: target2 target3  
2  
3 target1: dependency1 dependency2  
4     recipel  
5  
6 target2: target1  
7     recipe2  
8  
9 target3: dependency3  
10    recipe3
```

# Makefile

```
1 .PHONY: all
2 all: target2 target3
3
4 target1: dependency1 dependency2
5     recipe1
6
7 target2: target1
8     recipe2
9
10 target3: dependency3
11    recipe3
```

# Makefile



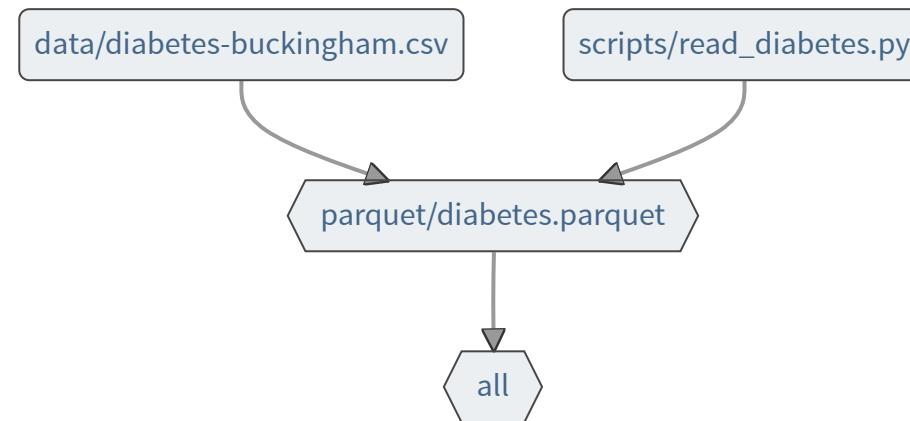
# Makefile: make clean

```
1 .PHONY: all clean
2 all: target2 target3
3
4 # --snip--
5
6 clean:
7   rm -f target1 target2 target3
```

## *Your Turn 2*

**Work through Your Turn 2 in `exercises.qmd`**

# Your Turn 2



# Re-running targets

```
1 make
```

```
1 make: `figures/gapminder_plot.png' is up to date.
```

scripts/create\_line\_plot.py

```
1 def create_line_plot(df: pl.DataFrame) -> plt.Figure:
2     ## -- snip --
3     g = sns.FacetGrid(
4         df,
5         col="continent",
6         col_wrap=2,
7         height=4,
8         sharey=True,
9         hue="country",
10        palette="mako"
11    )
12    ## -- snip --
13    return fig
14
15 ## -- snip --
```

# Outdated targets

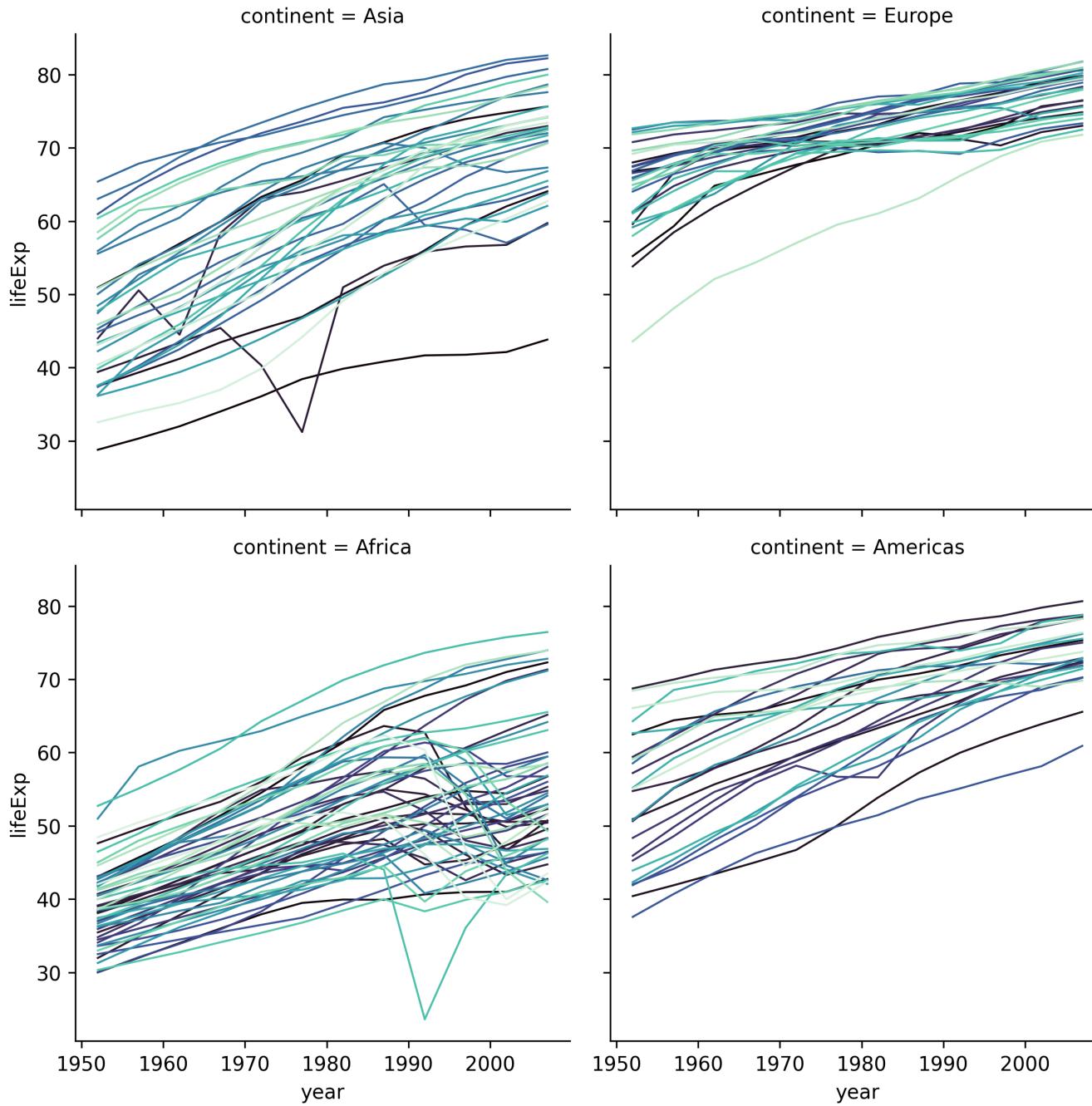
```
1 make --dry-run
```

```
1 uv run scripts/create_line_plot.py
```

# Outdated targets

```
1 make
```

```
uv run scripts/create_line_plot.py
```



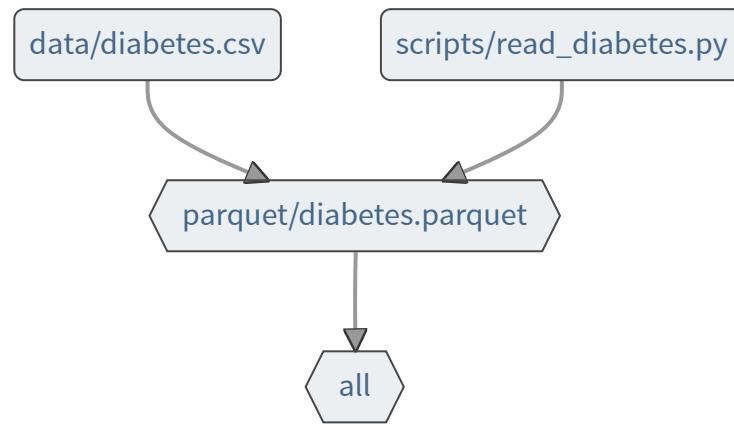
## *Your Turn 3*

Change the `parquet/diabetes.parquet` target to depend on `diabetes.csv` instead. Also modify `scripts/read_diabetes.py` to use this file.

Predict which targets are going to re-run, then run `make -n`. Were you right?

Run `make`

# Your Turn 3



# Building up your pipeline

- *Good targets are meaningful units of your analysis or important dependencies like files*
- *Add one or two targets at a time*
- *Run make -n and make often*

## ***Your Turn 4*** and ***Your Turn 5***

**Work through *Your Turn 4* and *Your Turn 5* in  
exercises.qmd**

TODO: add mermaid for 4 and 5

# Including Quarto files as targets: `report.qmd`

```
1  ```{python}
2 #| label: setup
3 #| include: false
4 import polars as pl
5 gapminder = pl.read_csv("data/gapminder.csv")
6
7
8 These data have `{{python}} len(gapminder)` observations.
9
10 ````{python}
11 #| label: fig-one
12 #| fig-cap: "Figure 1"
13 import IPython.display as disp
14 disp.Image("figures/gapminder_plot.png")
15 ````
```

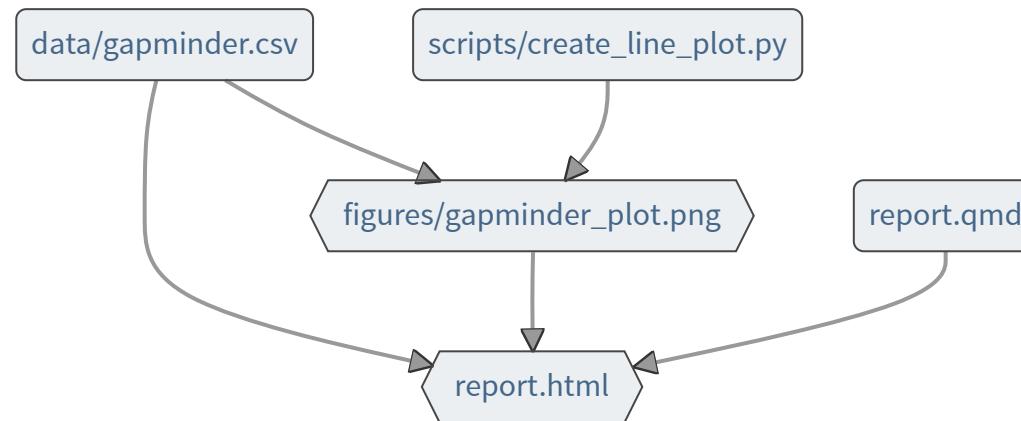
### Makefile

```
1 report.html: report.qmd figures/gapminder_plot.png data/gapminder.csv  
2     uv run quarto render report.qmd  
3  
4 figures/gapminder_plot.png: scripts/create_line_plot.py data/gapminder.csv  
5     uv run scripts/create_line_plot.py
```

# make

```
uv run quarto render report.qmd
```

# Including Quarto files as targets



## *Your Turn 6*

**Work through Your Turn 6 in `exercises.qmd`**

TODO: mermaid

# Removing target files

- `make clean`: A custom but common command to remove all created targets. Resets the pipeline to scratch.

## *Your Turn 7*

**Confirm that you can reproduce your entire pipeline from scratch. In the terminal:**

**Run `make clean`**

**Run `make -n`**

**Run `make`**

# Automatic parallelization

```
1 make --jobs=N
```

# Automatic parallelization

```
1 make --jobs=4
```

# Automatic parallelization

```
1 make -j4
```

# Wildcards

```
1 .PHONY: all clean
2
3 # Find all CSV files in the data/ directory.
4 CSV_FILES := $(wildcard data/*.csv)
5 # Generate a list of Parquet file names in the parquet/ directory.
6 PARQUET_FILES := $(patsubst data/%.csv,parquet/%.parquet,$(CSV_FILES))
7
8 # Default target: build all Parquet files.
9 all: $(PARQUET_FILES)
10
11 # Pattern rule: convert each CSV file to a Parquet file.
12 parquet/%.parquet: data/%.csv
13     uv run scripts/convert_csv_to_parquet.py $< $@
14
15 clean:
16     rm -f parquet/*.parquet
```

# Resources

[The GNU Make Manual](#)

[The Software Carpentry Make Course](#)