In [ ]:
```
import marimo as mo
```

In [ ]:
```
import random
import altair as alt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

# Set random seeds for reproducible snapshots
random.seed(42)
np.random.seed(42)
```

Kitchen Sink Notebook

This notebook demonstrates all major marimo features including:

- app.setup
- markdown
- app.function
- app.class_definition
- app.embed()
- altair charts
- plotly charts
- matplotlib charts
- matplotlib interactive plots

1. Markdown Examples

marimo supports rich markdown with **bold**, *italic*, and `code` formatting.

Lists

- Item 1
- Item 2
- Item 3

Code blocks

```python
def hello_world():
    print("Hello, marimo!")
```

Math

Inline math: $E = mc^2$

Block math:

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

2. app.function Example

Functions decorated with `@app.function` can be used throughout the notebook.

In [ ]:
```
def calculate_stats(data) -> dict[str, float]:
    """Calculate basic statistics from a numpy array."""
    return {
        "mean": data.mean(),
        "std": data.std(),
        "min": data.min(),
        "max": data.max(),
    }
```

In [ ]:
```
random.seed(42)
np.random.seed(42)
sample_data = np.random.randn(1000)
stats = calculate_stats(sample_data)
mo.md(f"""
### Statistics for random data:
- Mean: {stats["mean"]:.3f}
- Std Dev: {stats["std"]:.3f}
- Min: {stats["min"]:.3f}
- Max: {stats["max"]:.3f}
""")
```

Statistics for random data:

- Mean: 0.019
- Std Dev: 0.979
- Min: -3.241
- Max: 3.853

3. app.class_definition Example

Classes decorated with `@app.class_definition` can be instantiated and used across cells.

In [ ]:
```
class DataProcessor:
    """A sample class for processing data."""

    def __init__(self, data):
        self.data = data
        self.processed = False

    def process(self):
        """Apply some transformation to the data."""
        self.data = self.data * 2 + 10
        self.processed = True
        return self.data

    def get_summary(self):
        """Get a summary of the data."""
        return {
            "size": len(self.data),
            "processed": self.processed,
            "sum": self.data.sum(),
        }
```

In [ ]:
```
processor = DataProcessor(np.array([1, 2, 3, 4, 5]))
processed_data = processor.process()
summary = processor.get_summary()

mo.md(f"""
### DataProcessor Results:
- Processed data: {processed_data}
- Summary: {summary}
""")
```

DataProcessor Results:

- Processed data: [12 14 16 18 20]
- Summary: {'size': 5, 'processed': True, 'sum': np.int64(80)}

4. UI Elements

In [ ]:
```
# Create some interactive UI elements
slider = mo.ui.slider(1, 100, value=50, label="Select a value")
text_input = mo.ui.text(value="Hello, marimo!", placeholder="Enter text...")
dropdown = mo.ui.dropdown(
    options=["Option 1", "Option 2", "Option 3"],
    value="Option 1",
    label="Choose an option",
)

mo.vstack([mo.md("### Interactive UI Elements"), slider, text_input, dropdow
```

Interactive UI Elements

In [ ]:
```
mo.md(f"""
### Current UI Values:
- Slider: **{slider.value}**
- Text: **{text_input.value}**
- Dropdown: **{dropdown.value}**
""")
```

Current UI Values:

- Slider: **50**
- Text: **Hello, marimo!**
- Dropdown: **Option 1**

4b. `app.emebed()`

In [ ]:
```
from sub_notebook import app
```

In [ ]:
```
(await app.embed()).output
```

Sub Notebook

This is a sub notebook.

5. Altair Charts

Altair provides a declarative API for creating statistical visualizations.

In [ ]:
```
random.seed(42)
np.random.seed(42)
# Create sample data for Altair
altair_data = pd.DataFrame(
    {
        "x": np.arange(50),
        "y": np.cumsum(np.random.randn(50)),
        "category": np.random.choice(["A", "B", "C"], 50),
    }
)

# Create an Altair chart
altair_chart = (
    alt.Chart(altair_data)
    .mark_line(point=True)
    .encode(
        x=alt.X("x:Q", title="Time"),
        y=alt.Y("y:Q", title="Value"),
        color=alt.Color("category:N", title="Category"),
        tooltip=["x", "y", "category"],
    )
    .properties(
        width=600,
        height=400,
        title="Altair Line Chart with Multiple Categories",
    )
    .interactive()
)

altair_chart
```

In [ ]:
```
# Another Altair example - bar chart
bar_data = pd.DataFrame(
    {"category": ["A", "B", "C", "D", "E"], "value": [23, 45, 56, 34, 67]}
)

bar_chart = (
    alt.Chart(bar_data)
    .mark_bar()
    .encode(
        x=alt.X("category:N", title="Category"),
        y=alt.Y("value:Q", title="Value"),
        color=alt.Color("value:Q", scale=alt.Scale(scheme="viridis")),
        tooltip=["category", "value"],
    )
    .properties(width=600, height=300, title="Altair Bar Chart")
)

bar_chart
```

6. Plotly Charts

Plotly provides interactive, publication-quality graphs.

In [ ]:
```
random.seed(42)
np.random.seed(42)
# Create a Plotly scatter plot
x_data = np.random.randn(100)
y_data = np.random.randn(100)
color_data = np.random.randn(100)

plotly_scatter = go.Figure(
    data=go.Scatter(
        x=x_data,
        y=y_data,
        mode="markers",
        marker=dict(
            size=10,
            color=color_data,
            colorscale="Viridis",
            showscale=True,
            colorbar=dict(title="Value"),
        ),
        text=[f"Point {i}" for i in range(100)],
    )
)
```

```python
plotly_scatter.update_layout(
    title="Plotly Scatter Plot",
    xaxis_title="X Axis",
    yaxis_title="Y Axis",
    hovermode="closest",
    width=700,
    height=500,
)

plotly_scatter
```

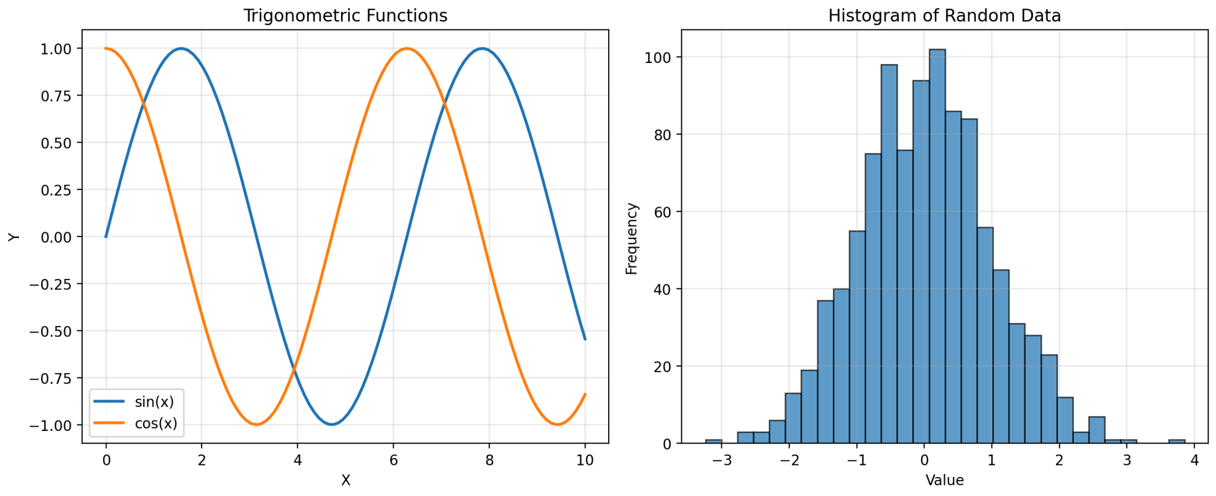
In [ ]:
```python
# Create a Plotly Express 3D scatter plot
t = np.linspace(0, 10, 100)
x_3d = np.sin(t)
y_3d = np.cos(t)
z_3d = t

plotly_3d = px.scatter_3d(
    x=x_3d,
    y=y_3d,
    z=z_3d,
    color=t,
    labels={"x": "X", "y": "Y", "z": "Z"},
    title="Plotly 3D Spiral",
    color_continuous_scale="Plasma",
)

plotly_3d.update_traces(marker=dict(size=5))
plotly_3d.update_layout(width=700, height=600)

plotly_3d
```

7. Matplotlib Charts

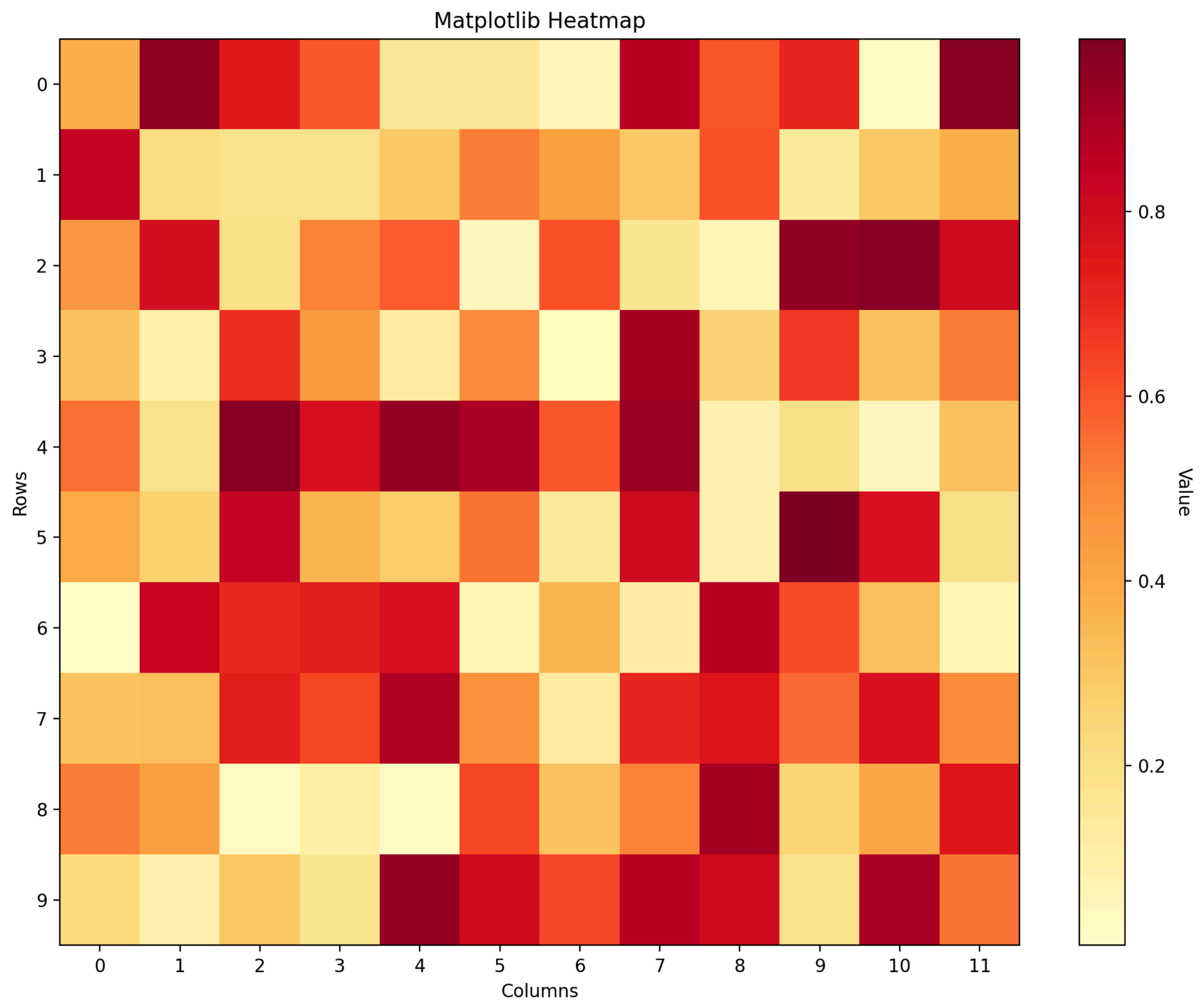
Matplotlib is the foundational plotting library for Python.

In [ ]:
```python
random.seed(42)
np.random.seed(42)
# Create a Matplotlib figure
fig1, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Line plot
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

ax1.plot(x, y1, label="sin(x)", linewidth=2)
ax1.plot(x, y2, label="cos(x)", linewidth=2)
ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_title("Trigonometric Functions")
ax1.legend()
```

```
ax1.grid(True, alpha=0.3)

# Histogram
data_hist = np.random.randn(1000)
ax2.hist(data_hist, bins=30, edgecolor="black", alpha=0.7)
ax2.set_xlabel("Value")
ax2.set_ylabel("Frequency")
ax2.set_title("Histogram of Random Data")
ax2.grid(True, alpha=0.3)

plt.tight_layout()
fig1
```



```
In [ ]: random.seed(42)
np.random.seed(42)
# Heatmap example
fig2, ax = plt.subplots(figsize=(10, 8))

data_matrix = np.random.rand(10, 12)
im = ax.imshow(data_matrix, cmap="YlOrRd", aspect="auto")

ax.set_xticks(np.arange(12))
ax.set_yticks(np.arange(10))
ax.set_xlabel("Columns")
ax.set_ylabel("Rows")
ax.set_title("Matplotlib Heatmap")

# Add colorbar
cbar = plt.colorbar(im, ax=ax)
cbar.set_label("Value", rotation=270, labelpad=15)

plt.tight_layout()
fig2
```



8. Matplotlib Interactive

Create interactive matplotlib plots that respond to UI elements.

```
In [ ]: # Create interactive matplotlib plot
fig_interactive, ax_interactive = plt.subplots(figsize=(12, 6))

x_interactive = np.linspace(0, 10, 500)
_amplitude = 1.0
_freq = 1.0
_phase = 0.0
y_interactive = _amplitude * np.sin(_freq * x_interactive + _phase)

ax_interactive.plot(x_interactive, y_interactive, linewidth=2, color="blue")

ax_interactive.set_xlabel("X", fontsize=12)
ax_interactive.set_ylabel("Y", fontsize=12)
ax_interactive.set_title(
    f"Interactive Plot: A={_amplitude:.1f}, f={_freq:.1f}, φ={_phase:.2f}",
    fontsize=14,
)
ax_interactive.grid(True, alpha=0.3)
ax_interactive.axhline(y=0, color="k", linestyle="-", linewidth=0.5)
ax_interactive.axvline(x=0, color="k", linestyle="-", linewidth=0.5)

plt.tight_layout()
mo.mpl.interactive(fig_interactive)
```

9. Images

Display images from local files and remote URLs.

Using HTML `<img>` tags

Local cat image

Using `mo.image()`

In [ ]:
```
# Local image using mo.image()
local_image = mo.image(
    src=mo.notebook_dir() / "public/cat.jpg", width=400, alt="Local cat imag
)

# Remote image using mo.image()
remote_image = mo.image(
    src="https://raw.githubusercontent.com/marimo-team/marimo/main/docs/_sta
    width=400,
    alt="Marimo logo from remote URL",
)

mo.hstack(
    [
        mo.md("**Local image:**"),
        local_image,
        mo.md("**Remote image:**"),
        remote_image,
    ]
)
```

Local image:

Remote image: