Strategy Lab — Step-by-Step Build Guide

Audience: MScFMI students with little or no programming background. **Goal:** Build a Python backtesting framework from scratch, one file at a time, finishing with an interactive Streamlit dashboard and a downloadable PDF report.

Prerequisites

Tool	Why you need it
uv	Ultra-fast Python & package manager
Google Antigravity	AI-powered code editor & orchestrator
Claude Opus 4.6	AI coding agent inside Antigravity

[!TIP] You do **not** need to install Python separately — `uv` will download and manage the correct Python version for you automatically.

Step 0 — Understand the Folder Layout

When you are done, your project will look like this:

```
project/
  pyproject.toml            ← Package metadata & dependencies
  src/
      strategy_lab/
          __init__.py       ← Makes this folder a Python package
          data.py           ← Data loading & cleaning
          engine.py         ← Backtesting engine
          metrics.py        ← Performance metrics
          report_builder.py ← PDF report generator
          app/
              __init__.py
              main.py       ← Streamlit dashboard
  tests/
      test_data.py
      test_engine.py
      test_metrics.py
      test_integration.py
```

You will create **every file listed above**, step by step.

Step 1 — Install uv

1. Open the **Antigravity terminal** (click the terminal icon at the bottom of the window, or press `Ctrl+`).
2. Install **uv** by running the appropriate command for your system:

   Windows (PowerShell):

   ```
   powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
   ```

Mac / Linux:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

3. Close and re-open your terminal, then verify the installation:

   ```
   uv --version
   ```

   You should see a version number (e.g. `uv 0.6.x`).

   [!TIP] Unlike Anaconda, `uv` does not require you to "activate" anything. It automatically picks up the project's Python version and dependencies from `pyproject.toml` every time you run `uv run ...`.

Step 2 — Create the Package Metadata (`pyproject.toml`)

In the `project/` folder, create a file called **`pyproject.toml`** and paste:

> [!TIP] You can ask **Claude** to create this file for you! In the Antigravity chat, type: *"Create a pyproject.toml for a package called strategy_lab with pandas, numpy, scipy, streamlit, plotly, and reportlab as dependencies."*

```toml
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"

[project]
name = "strategy_lab"
version = "0.1.0"
description = "A backtesting framework for the Computational Investing course."
requires-python = ">=3.10"
dependencies = [
    "pandas",
    "numpy",
    "scipy",
    "streamlit",
    "plotly",
    "reportlab",
    "yfinance",
    "matplotlib",
]

[project.optional-dependencies]
dev = [
    "pytest",
]
```

Then install the package in **editable mode** (this lets Python find your code):

```
uv pip install -e ".[dev]"
```

> [!NOTE] - The `-e` flag means *editable*. Any changes you save in the source files will immediately take effect — no need to re-install. - The `[dev]` part also installs `pytest` for testing. - `uv` will automatically create a virtual environment for you the first time.

Step 3 — Create the Package Skeleton

You can ask **Claude** to do this for you: *"Create the folder structure for the strategy_lab package with an app sub-package and a tests folder."*

Or do it manually in the terminal:

```
mkdir -p src/strategy_lab/app
mkdir -p tests
```

(On Windows PowerShell, use `New-Item -ItemType Directory -Path src\strategy_lab\app, tests -Force`)

Then create two empty `__init__.py` files — these tell Python that the folders are packages:

File to create	Content
`src/strategy_lab/__init__.py`	*(leave the file empty)*
`src/strategy_lab/app/__init__.py`	*(leave the file empty)*

Step 4 — Build `data.py` (Data Loading & Cleaning)

Create the file **`src/strategy_lab/data.py`**.

4.1 — Imports

```
import pandas as pd
import numpy as np
```

4.2 — `validate_data` function

This function checks that the data is suitable for backtesting.

```
def validate_data(df: pd.DataFrame, silent: bool = False) -> pd.DataFrame:
    """
    Checks the validity of the input DataFrame for backtesting.

    Checks:
    1. Index is a DatetimeIndex and is monotonic increasing.
    2. Checks for NaN values and reports them.
    """
    # Make sure the index contains dates
    if not isinstance(df.index, pd.DatetimeIndex):
        try:
            df.index = pd.to_datetime(df.index)
        except Exception as e:
            raise ValueError(f"Index could not be converted to DatetimeIndex: {e}")

    # Sort by date if not already sorted
    if not df.index.is_monotonic_increasing:
        df = df.sort_index()

    # Warn about missing values
    if not silent and df.isnull().values.any():
        print("Warning: Input data contains NaN values. Use clean_data to handle them.")
```

```
    return df
```

What is happening?

- We need dates as the index so we can reason about time.
- The data must be sorted oldest → newest (monotonic increasing).
- We warn (but do not crash) if there are missing values.

4.3 — `clean_data` function

```
def clean_data(df: pd.DataFrame, method='ffill') -> pd.DataFrame:
    """
    Cleans the data by handling missing values.

    Args:
        method: 'ffill' (forward-fill), 'bfill' (backward-fill),
                or 'fill_zero'.
    """
    if method == 'ffill':
        return df.ffill()
    elif method == 'bfill':
        return df.bfill()
    elif method == 'fill_zero':
        return df.fillna(0.0)
    else:
        raise ValueError(f"Unknown cleaning method: {method}")
```

What is happening?

- Forward-fill (`ffill`) replaces a missing value with the last known value. This is the safest default for financial time series.

4.4 — `DataLoader` class

```
class DataLoader:
    """Helper class to load data from CSV."""

    @staticmethod
    def load_csv(filepath: str, date_col: str = 'Date') -> pd.DataFrame:
        df = pd.read_csv(filepath, parse_dates=[date_col], index_col=date_col)
        return validate_data(df, silent=True)
```

4.5 — `download_data` function

```
def download_data(tickers: list[str], start_date: str, end_date: str = None) -> pd.DataFrame:
    """Downloads historical price data using yfinance."""
    import yfinance as yf

    if isinstance(tickers, str):
        tickers = [t.strip() for t in tickers.split(',')]

    data = yf.download(
        tickers,
        start=start_date,
```

```
        end=end_date,
        auto_adjust=True,
        progress=False,
        group_by='column'
    )

    if len(tickers) == 1:
        if 'Close' in data.columns:
            data = data[['Close']]
            data.columns = tickers
    else:
        if 'Close' in data.columns:
            data = data['Close']

    return data
```

4.6 — Helper functions

```
def price_to_log_returns(df: pd.DataFrame) -> pd.DataFrame:
    """Converts prices to log returns: ln(P_t / P_{t-1})."""
    df = df.astype(float)
    return np.log(df / df.shift(1))

def resample_to_monthly(df: pd.DataFrame) -> pd.DataFrame:
    """Resamples daily data to monthly (last price of each month)."""
    return df.resample('ME').last()
```

> [!NOTE] Log returns are additive over time, which makes cumulative return calculations straightforward: just sum them up.

Checkpoint — Test `data.py`

Create the folder `tests/` and a file **`tests/test_data.py`**:

```
import pytest
import pandas as pd
import numpy as np
from strategy_lab.data import validate_data, clean_data

def test_validate_data_valid():
    dates = pd.date_range(start='2020-01-01', periods=5, freq='M')
    df = pd.DataFrame(np.random.randn(5, 2), index=dates, columns=['A', 'B'])
    validated_df = validate_data(df)
    assert validated_df.shape == (5, 2)
    assert validated_df.index.is_monotonic_increasing

def test_validate_data_invalid_index():
    df = pd.DataFrame({'A': [1, 2, 3]}, index=['a', 'b', 'c'])
    with pytest.raises(ValueError):
        validate_data(df)

def test_clean_data_ffill():
    dates = pd.date_range(start='2020-01-01', periods=5, freq='M')
    df = pd.DataFrame({'A': [1, np.nan, 3, np.nan, 5]}, index=dates)
```

```
    cleaned = clean_data(df, method='ffill')
    assert cleaned['A'].isnull().sum() == 0
    assert cleaned.iloc[1]['A'] == 1.0
```

Run the tests:

```
uv run pytest tests/test_data.py -v
```

You should see **3 passed** .

Step 5 — Build `metrics.py` (Performance Metrics)

Create **`src/strategy_lab/metrics.py`**.

5.1 — Imports

```
import numpy as np
import pandas as pd
```

5.2 — Cumulative returns and drawdown

```
def calculate_cumulative_returns(returns: pd.Series) -> pd.Series:
    """Cumulative wealth from log returns: exp(cumsum(log_returns))."""
    return np.exp(returns.cumsum())

def calculate_drawdown(returns: pd.Series) -> pd.DataFrame:
    """Calculates wealth, peaks, and drawdown from log returns."""
    wealth = np.exp(returns.cumsum())
    peaks = wealth.cummax()
    drawdown = (wealth - peaks) / peaks
    return pd.DataFrame({'Wealth': wealth, 'Peaks': peaks, 'Drawdown': drawdown})
```

What is a drawdown?

A drawdown measures how far a portfolio has fallen from its highest peak. A drawdown of -20 % means the portfolio has lost 20 % from its best level.

5.3 — Risk-adjusted ratios

```
def sharpe_ratio(returns: pd.Series, risk_free_rate: float = 0.0,
                 periods_per_year: int = 12) -> float:
    """Annualized Sharpe Ratio."""
    excess_returns = (returns - risk_free_rate) / periods_per_year
    if excess_returns.std() == 0:
        return 0.0
    return np.sqrt(periods_per_year) * excess_returns.mean() / excess_returns.std()

def sortino_ratio(returns: pd.Series, risk_free_rate: float = 0.0,
                  periods_per_year: int = 12) -> float:
    """Annualized Sortino Ratio (penalises only downside volatility)."""
    excess_returns = returns - risk_free_rate / periods_per_year
    downside_returns = excess_returns[excess_returns < 0]
```

```
    if len(downside_returns) == 0 or downside_returns.std() == 0:
        return np.inf

    downside_deviation = np.sqrt(np.mean(downside_returns**2))
    if downside_deviation == 0:
        return np.inf

    return np.sqrt(periods_per_year) * excess_returns.mean() / downside_deviation

def calmar_ratio(returns: pd.Series, periods_per_year: int = 12) -> float:
    """Calmar Ratio = Annualised Return / Max Drawdown."""
    wealth = np.exp(returns.cumsum())
    peaks = wealth.cummax()
    drawdown = (wealth - peaks) / peaks
    max_drawdown = drawdown.min()

    if max_drawdown == 0:
        return np.inf

    annualized_ret = np.exp(returns.mean() * periods_per_year) - 1
    return annualized_ret / abs(max_drawdown)
```

5.4 — Additional statistics

```
def calculate_turnover(weights: pd.DataFrame) -> pd.Series:
    """Portfolio turnover = sum of absolute weight changes."""
    return weights.diff().abs().sum(axis=1)

def annualized_return(returns: pd.Series, periods_per_year: int = 12) -> float:
    """Annualised return from log returns."""
    if len(returns) == 0:
        return 0.0
    return np.exp(returns.mean() * periods_per_year) - 1

def annualized_volatility(returns: pd.Series, periods_per_year: int = 12) -> float:
    """Annualised volatility."""
    return returns.std() * np.sqrt(periods_per_year)

def skewness(returns: pd.Series) -> float:
    """Skewness of the return distribution."""
    return returns.skew()

def excess_kurtosis(returns: pd.Series) -> float:
    """Excess kurtosis (normal distribution = 0)."""
    return returns.kurtosis()
```

Checkpoint — Test `metrics.py`

Create **`tests/test_metrics.py`**:

```
import pytest
import pandas as pd
import numpy as np
from strategy_lab.metrics import sharpe_ratio, sortino_ratio, calculate_drawdown
```

```
def test_sharpe_ratio():
    returns = pd.Series([0.01, 0.01, 0.01, 0.01])
    # Std dev is 0 → Sharpe 0
    assert sharpe_ratio(returns) == 0.0

    returns_2 = pd.Series([0.1, -0.1, 0.1, -0.1])
    # Mean 0 → Sharpe 0
    assert sharpe_ratio(returns_2) == 0.0

def test_drawdown():
    returns = pd.Series([0.1, -0.1, 0.212121])
    dd_df = calculate_drawdown(returns)
    drawdown = dd_df['Drawdown']

    assert np.isclose(drawdown.iloc[0], 0.0)
    assert np.isclose(drawdown.iloc[1], -0.1)
    assert np.isclose(drawdown.iloc[2], 0.0)
```

```
uv run pytest tests/test_metrics.py -v
```

You should see **2 passed** .

Step 6 — Build `engine.py` (Backtesting Engine)

Create **`src/strategy_lab/engine.py`**.

This is the heart of the project. The engine takes a *strategy function* (that you define) and applies it in a walk-forward fashion: train on the past, predict weights for the next period, move forward, repeat.

```
import pandas as pd
import numpy as np
from typing import Callable
from .data import validate_data, clean_data

class BacktestEngine:
    """
    Runs a walk-forward backtest on a monthly return series.

    Attributes:
        data: DataFrame with DatetimeIndex and one column per asset.
    """

    def __init__(self, data: pd.DataFrame):
        self.data = validate_data(data, silent=True)

    def run(self,
            strategy_func: Callable[[pd.DataFrame], pd.Series],
            train_window_months: int,
            test_window_months: int,
            window_type: str = 'rolling') -> pd.DataFrame:
        """
        Args:
            strategy_func: function(history_df) → pd.Series of weights.
```

```
        train_window_months: months of history to feed the strategy.
        test_window_months: months to hold positions before rebalancing.
        window_type: 'rolling' (fixed-size window) or 'expanding'.

    Returns:
        DataFrame with a 'Strategy' column (returns) and one column
        per asset (the weights used in each period).
    """
    if window_type not in ['rolling', 'expanding']:
        raise ValueError("window_type must be 'rolling' or 'expanding'")

    data = self.data
    n_rows = len(data)

    # Detect frequency
    freq = pd.infer_freq(data.index)
    if freq is None:
        days_diff = (data.index[1] - data.index[0]).days
        if not (28 <= days_diff <= 31):
            print("Warning: Could not infer frequency. "
                  "Assuming data is Monthly.")

    train_steps = train_window_months
    test_steps = test_window_months

    strategy_returns = []
    weights_history = []
    current_step = train_steps

    while current_step + test_steps <= n_rows:
        # ---- Training window ----
        if window_type == 'rolling':
            train_start = current_step - train_steps
        else:  # expanding
            train_start = 0

        train_data = data.iloc[train_start:current_step]

        # ---- Compute weights ----
        try:
            weights = strategy_func(train_data)
        except Exception as e:
            print(f"Error in strategy at step {current_step}: {e}")
            weights = pd.Series(0, index=data.columns)

        weights = weights.reindex(data.columns, fill_value=0)

        # ---- Apply to test window ----
        test_end = min(current_step + test_steps, n_rows)
        test_returns = data.iloc[current_step:test_end]
        portfolio_return = (test_returns * weights).sum(axis=1)

        strategy_returns.append(portfolio_return)
```

```
            for _ in range(len(test_returns)):
                weights_history.append(weights)

            current_step += test_steps

        if not strategy_returns:
            return pd.DataFrame()

        full_returns = pd.concat(strategy_returns)
        full_weights = pd.DataFrame(weights_history, index=full_returns.index)

        result = pd.concat([full_returns.rename('Strategy'), full_weights], axis=1)
        return result
```

Key concepts for non-CS students

Concept	Plain-English Explanation
Walk-forward	We never look into the future. The strategy only sees data *before* the test period.
Rolling window	A fixed-size sliding window (e.g. always the last 12 months).
Expanding window	Grows over time — uses *all* data from the beginning up to the current date.
Rebalancing	Every `test_window_months` months the weights are recalculated.

Checkpoint — Test `engine.py`

Create **`tests/test_engine.py`**:

```
import pandas as pd
import numpy as np
from strategy_lab.engine import BacktestEngine

def test_backtest_rolling_window():
    dates = pd.date_range(start='2020-01-01', periods=20, freq='M')
    data = pd.DataFrame(
        np.ones((20, 2)) * 0.01,
        index=dates, columns=['A', 'B']
    )

    def strategy(hist_data):
        return pd.Series([0.5, 0.5], index=hist_data.columns)

    engine = BacktestEngine(data)
    results = engine.run(strategy, train_window_months=12,
                         test_window_months=1, window_type='rolling')

    assert len(results) == 8
    assert 'Strategy' in results.columns
    assert np.allclose(results['Strategy'], 0.01)

def test_backtest_expanding_window():
    dates = pd.date_range(start='2020-01-01', periods=20, freq='M')
    data = pd.DataFrame(np.random.randn(20, 2), index=dates, columns=['A', 'B'])
```

```
    calls = []
    def strategy(hist_data):
        calls.append(len(hist_data))
        return pd.Series([0.5, 0.5], index=hist_data.columns)

    engine = BacktestEngine(data)
    engine.run(strategy, train_window_months=10,
               test_window_months=1, window_type='expanding')

    assert calls[0] == 10   # first call sees 10 rows
    assert calls[1] == 11   # second call sees 11 rows (expanding)
```

```
uv run pytest tests/test_engine.py -v
```

You should see **2 passed** .

Step 7 — Build `report_builder.py` (PDF Export)

Create **`src/strategy_lab/report_builder.py`**.

```
from reportlab.lib.pagesizes import letter
from reportlab.lib import colors
from reportlab.platypus import (SimpleDocTemplate, Table, TableStyle,
                                Paragraph, Spacer, Image)
from reportlab.lib.styles import getSampleStyleSheet
import pandas as pd
import numpy as np
import io
import matplotlib.pyplot as plt


def generate_pdf_report(
    results_df: pd.DataFrame,
    metrics_df: pd.DataFrame,
    filename: str = "backtest_report.pdf"
):
    """Generates a PDF with metrics table, cumulative-return plot, and drawdown plot."""

    doc = SimpleDocTemplate(filename, pagesize=letter)
    styles = getSampleStyleSheet()
    story = []

    # ---- Title ----
    story.append(Paragraph("Backtest Report", styles['Title']))
    story.append(Spacer(1, 12))

    # ---- Metrics Table ----
    story.append(Paragraph("Performance Metrics", styles['Heading2']))

    data = [['Metric', 'Value']]
    for idx, row in metrics_df.iterrows():
        val = row.iloc[0]
        if isinstance(val, float):
```

```
            val = f"{val:.4f}"
        data.append([idx, val])

    t = Table(data)
    t.setStyle(TableStyle([
        ('BACKGROUND', (0, 0), (-1, 0), colors.grey),
        ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
        ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
        ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
        ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
        ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
        ('GRID', (0, 0), (-1, -1), 1, colors.black),
    ]))
    story.append(t)
    story.append(Spacer(1, 24))

    # ---- Cumulative Returns Plot ----
    story.append(Paragraph("Cumulative Returns", styles['Heading2']))
    plt.figure(figsize=(6, 4))
    plt.plot((1 + results_df['Strategy']).cumprod(), label='Strategy')
    plt.title("Cumulative Returns")
    plt.legend()
    plt.grid(True)
    buf1 = io.BytesIO()
    plt.savefig(buf1, format='png')
    plt.close()
    buf1.seek(0)
    story.append(Image(buf1, width=400, height=300))
    story.append(Spacer(1, 12))

    # ---- Drawdown Plot ----
    story.append(Paragraph("Drawdown", styles['Heading2']))
    plt.figure(figsize=(6, 4))
    wealth = (1 + results_df['Strategy']).cumprod()
    dd = (wealth - wealth.cummax()) / wealth.cummax()
    plt.plot(dd, label='Drawdown', color='red')
    plt.title("Drawdown")
    plt.legend()
    plt.grid(True)
    buf2 = io.BytesIO()
    plt.savefig(buf2, format='png')
    plt.close()
    buf2.seek(0)
    story.append(Image(buf2, width=400, height=300))

    doc.build(story)
    return filename
```

Step 8 — Build the Streamlit Dashboard (`app/main.py`)

Create `src/strategy_lab/app/main.py`.

This is the interactive front-end for your backtesting framework.

8.1 — Imports and page config

```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px
import sys, os

# Allow imports when running as a script
current_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.dirname(os.path.dirname(current_dir))
if parent_dir not in sys.path:
    sys.path.append(parent_dir)

from strategy_lab.data import (validate_data, clean_data, download_data,
                               price_to_log_returns, resample_to_monthly)
from strategy_lab.engine import BacktestEngine
from strategy_lab.metrics import (calculate_cumulative_returns, calculate_drawdown,
                                  sharpe_ratio, sortino_ratio, calmar_ratio,
                                  annualized_return, annualized_volatility,
                                  skewness, excess_kurtosis)
from strategy_lab.report_builder import generate_pdf_report

st.set_page_config(page_title="Strategy Assessment Lab", layout="wide")
st.title("Strategy Assessment Lab")
st.markdown("### Computational Investing - Strategy Backtesting")
```

8.2 — Sidebar: Data Input

```
st.sidebar.header("1. Input Data")
data_source = st.sidebar.radio("Data Source", ["Upload CSV", "Download (yfinance)"])

df = None

if data_source == "Upload CSV":
    uploaded_data = st.sidebar.file_uploader("Upload Market Data (CSV)", type=['csv'])
    if uploaded_data:
        try:
            df = pd.read_csv(uploaded_data, index_col=0, parse_dates=True)
            st.sidebar.success("CSV Loaded")
        except Exception as e:
            st.sidebar.error(f"Error loading CSV: {e}")

elif data_source == "Download (yfinance)":
    tickers_input = st.sidebar.text_input("Tickers (comma separated)", "SPY, AGG, GLD")
    start_date = st.sidebar.date_input("Start Date", pd.to_datetime("2020-01-01"))
    end_date = st.sidebar.date_input("End Date", pd.to_datetime("today"))

    if st.sidebar.button("Fetch Data"):
        if tickers_input:
            with st.spinner("Downloading data..."):
```

```
                try:
                    df = download_data(tickers_input,
                                       start_date=str(start_date),
                                       end_date=str(end_date))
                    if df.empty:
                        st.sidebar.warning("No data found for tickers.")
                    else:
                        st.sidebar.success("Data Downloaded")
                        st.session_state['data'] = df
                except Exception as e:
                    st.sidebar.error(f"Download Error: {e}")

if data_source == "Download (yfinance)" and 'data' in st.session_state:
    df = st.session_state['data']
```

8.3 — Sidebar: Strategy & Parameters

```
st.sidebar.header("2. Strategy Configuration")
strategy_source = st.sidebar.radio("Strategy Source", ["Default (1/N)", "Upload Script"])

strategy_func = None

if strategy_source == "Upload Script":
    uploaded_script = st.sidebar.file_uploader("Upload Strategy Script (.py)", type=['py'])
    st.sidebar.info("Script must contain a function `get_weights(df)` returning a Series.")
    if uploaded_script:
        script_content = uploaded_script.read().decode("utf-8")
        local_scope = {}
        try:
            exec(script_content, globals(), local_scope)
            if 'get_weights' in local_scope:
                strategy_func = local_scope['get_weights']
                st.sidebar.success("Strategy loaded: get_weights")
            else:
                st.sidebar.error("Function `get_weights(df)` not found in script.")
        except Exception as e:
            st.sidebar.error(f"Error loading script: {e}")
else:
    def equal_weight_strategy(df):
        n_assets = len(df.columns)
        return pd.Series(1 / n_assets, index=df.columns)

    strategy_func = equal_weight_strategy
    st.sidebar.success("Using Default 1/N Strategy")

st.sidebar.header("3. Backtest Parameters")
train_window = st.sidebar.number_input("Training Window (Months)", min_value=1, value=12)
test_window = st.sidebar.number_input("Test Window (Months)", min_value=1, value=1)
window_type = st.sidebar.selectbox("Window Type", ["rolling", "expanding"])
```

8.4 — Main area: Run backtest & display results

```
if df is not None and strategy_func is not None:
    try:
        st.subheader("Data Preview (Prices)")
        st.dataframe(df.head())

        # Clean data
        if df.isnull().values.any():
            st.warning("Data contains NaNs. Filling with ffill.")
            df = clean_data(df, 'ffill')

        # Resample
        st.markdown("**Data Frequency Processing**")
        resample_monthly = st.checkbox("Resample to Monthly (End of Month)", value=True)
        if resample_monthly:
            df = resample_to_monthly(df)
            st.info("Data resampled to Monthly frequency.")

        # Convert to log returns
        st.markdown("**Converting Prices to Log Returns for Backtest**")
        returns_df = price_to_log_returns(df).dropna()

        st.subheader("Data Preview (Log Returns)")
        st.dataframe(returns_df.head())

        if st.button("Run Backtest"):
            with st.spinner("Running Backtest on Log Returns..."):
                engine = BacktestEngine(returns_df)
                results = engine.run(
                    strategy_func=strategy_func,
                    train_window_months=train_window,
                    test_window_months=test_window,
                    window_type=window_type
                )

                if results.empty:
                    st.error("Backtest returned no results. Check windows and data size.")
                    st.session_state['backtest_results'] = None
                else:
                    st.success("Backtest Complete!")
                    st.session_state['backtest_results'] = results

        # Show results if available
        if 'backtest_results' in st.session_state and st.session_state['backtest_results'] is not None:
            results = st.session_state['backtest_results']

            st.divider()
            col1, col2 = st.columns([2, 1])

            with col1:
                cum_ret = calculate_cumulative_returns(results['Strategy'])
                fig_cum = px.line(cum_ret, title="Cumulative Returns")
                st.plotly_chart(fig_cum, width='stretch')
```

```
            dd_df = calculate_drawdown(results['Strategy'])
            fig_dd = px.area(dd_df, x=dd_df.index, y='Drawdown', title="Drawdown")
            st.plotly_chart(fig_dd, width='stretch')

            weights = results.drop(columns=['Strategy'])
            fig_weights = px.area(weights, title="Asset Allocation")
            st.plotly_chart(fig_weights, width='stretch')

        with col2:
            sharpe = sharpe_ratio(results['Strategy'])
            sortino = sortino_ratio(results['Strategy'])
            calmar = calmar_ratio(results['Strategy'])
            total_ret = cum_ret.iloc[-1] - 1
            max_dd = dd_df['Drawdown'].min()
            ann_ret = annualized_return(results['Strategy'])
            ann_vol = annualized_volatility(results['Strategy'])
            skew = skewness(results['Strategy'])
            ex_kurt = excess_kurtosis(results['Strategy'])

            metrics_data = {
                "Metric": [
                    "Total Return", "Annualized Return",
                    "Annualized Volatility", "Max Drawdown",
                    "Sharpe Ratio", "Sortino Ratio",
                    "Calmar Ratio", "Skewness", "Excess Kurtosis"
                ],
                "Value": [
                    f"{total_ret:.1%}", f"{ann_ret:.1%}",
                    f"{ann_vol:.1%}", f"{max_dd:.1%}",
                    f"{sharpe:.1f}", f"{sortino:.1f}",
                    f"{calmar:.1f}", f"{skew:.1f}", f"{ex_kurt:.1f}"
                ]
            }
            metrics_df = pd.DataFrame(metrics_data).set_index("Metric")
            st.table(metrics_df)

            # PDF report
            st.subheader("Report Export")
            report_path = "backtest_report.pdf"
            if st.button("Generate PDF Report"):
                try:
                    generate_pdf_report(results, metrics_df, filename=report_path)
                    with open(report_path, "rb") as f:
                        pdf_data = f.read()
                    st.download_button(
                        label="Download PDF",
                        data=pdf_data,
                        file_name="strategy_report.pdf",
                        mime="application/pdf"
                    )
                    st.success("PDF Generated!")
                except Exception as e:
                    st.error(f"Error generating PDF: {e}")
```

```
    except Exception as e:
        st.error(f"An error occurred: {e}")
else:
    st.info("Please select a data source, provide data, and ensure a strategy is selected to proceed.")
```

Step 9 — Run the Integration Tests

Create **tests/test_integration.py**:

```
import pytest
import pandas as pd
import numpy as np
import os
from strategy_lab.engine import BacktestEngine
from strategy_lab.metrics import sharpe_ratio

reportlab = pytest.importorskip("reportlab")
from strategy_lab.report_builder import generate_pdf_report

def test_full_backtest_flow(tmp_path):
    # 1. Create synthetic data
    dates = pd.date_range(start='2020-01-01', periods=24, freq='M')
    data = pd.DataFrame(
        np.random.normal(0.01, 0.05, (24, 3)),
        index=dates, columns=['A', 'B', 'C']
    )

    # 2. Define a strategy (Inverse Volatility)
    def inverse_vol_strategy(history):
        vol = history.std()
        inv_vol = 1 / vol
        return inv_vol / inv_vol.sum()

    # 3. Run engine
    engine = BacktestEngine(data)
    results = engine.run(inverse_vol_strategy,
                         train_window_months=12,
                         test_window_months=1)

    assert not results.empty
    assert 'Strategy' in results.columns
    assert len(results.columns) == 4  # Strategy + 3 assets

    # 4. Generate PDF
    metrics_df = pd.DataFrame({'Value': [0.5, 0.1]}, index=['Sharpe', 'Drawdown'])
    report_file = tmp_path / "test_report.pdf"
    generate_pdf_report(results, metrics_df, filename=str(report_file))

    assert os.path.exists(report_file)
    assert os.path.getsize(report_file) > 0
```

Run all tests at once:

```
uv run pytest tests/ -v
```

You should see **7 passed** .

Step 10 — Launch the Dashboard

```
uv run streamlit run src/strategy_lab/app/main.py
```

A browser tab will open at `http://localhost:8501`.

Quick walkthrough:

1. In the sidebar, choose **"Download (yfinance)"**.
2. Type `SPY, AGG, GLD` and click **Fetch Data**.
3. Leave "Resample to Monthly" checked.
4. Click **Run Backtest**.
5. Explore the cumulative returns, drawdown, and asset allocation charts.
6. Click **Generate PDF Report** to download your results.

Troubleshooting

Problem	Solution
`ModuleNotFoundError: No module named 'strategy_lab'`	Run `uv pip install -e ".[dev]"` from the `project/` folder
`uv: command not found`	Close and re-open your terminal after installing uv
`yfinance` download fails	Check your internet connection; Yahoo Finance may be temporarily down
Tests fail with import errors	Run tests with `uv run pytest` so the correct environment is used
Streamlit shows a blank page	Check the terminal for error messages

---

Writing Your Own Strategy

To write a custom strategy, create a `.py` file with a function called `get_weights`:

```
import pandas as pd

def get_weights(df: pd.DataFrame) -> pd.Series:
    """
    Minimum variance weights (simplified).

    Args:
        df: historical returns (training window).

    Returns:
        pd.Series of weights summing to 1.
    """
    cov = df.cov()
```

```python
    inv_cov = pd.DataFrame(
        np.linalg.inv(cov.values),
        index=cov.index, columns=cov.columns
    )
    ones = pd.Series(1, index=cov.index)
    weights = inv_cov @ ones
    weights = weights / weights.sum()
    return weights
```

Upload this file through the dashboard sidebar under **"Upload Script"**.

Tips for Working with Claude in Antigravity

Claude Opus 4.6 is your AI pair-programmer inside Antigravity. Here are some effective ways to use it:

What you want	What to type in the chat
Create a file	*"Create the file `src/strategy_lab/data.py` with the validate_data and clean_data functions from Step 4"*
Explain code	*"Explain what the `portfolio_variance` function does in plain English"*
Debug an error	Paste the error message and ask *"What does this error mean and how do I fix it?"*
Run tests	*"Run the tests in `tests/test_data.py` and tell me what passed"*
Write a strategy	*"Write a get_weights function that implements inverse-volatility weighting"*

> [!IMPORTANT] Claude can write and edit files directly. If you are unsure about a step, **copy-paste the step description into the chat** and Claude will do it for you. Always review what it wrote before moving on.

You're done! You now have a working backtesting framework with:

- Data loading and cleaning
- Walk-forward backtesting engine
- Performance metrics (Sharpe, Sortino, Calmar, drawdown, …)
- Interactive Streamlit dashboard
- PDF report export
- A full test suite