automating-execution-jupyter-notebook-files-python-scripts-hugo-static-site-generation

June 29, 2025

1 Automating Execution of Jupyter Notebook Files, Python Scripts, and Hugo Static Site Generation

1.1 Python Imports

[1]:
```
# Standard Library
import datetime
import io
import os
import random
import sys
import warnings

from datetime import datetime, timedelta
from pathlib import Path

# Data Handling
import numpy as np
import pandas as pd

# Data Visualization
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import seaborn as sns
from matplotlib.ticker import FormatStrFormatter, FuncFormatter, MultipleLocator

# Data Sources
import yfinance as yf

# Statistical Analysis
import statsmodels.api as sm

# Machine Learning
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
# Suppress warnings
warnings.filterwarnings("ignore")
```

1.2 Add Directories To Path

[2]:
```
# Add the source subdirectory to the system path to allow import config from␣
 ↪settings.py
current_directory = Path(os.getcwd())
website_base_directory = current_directory.parent.parent.parent
src_directory = website_base_directory / "src"
sys.path.append(str(src_directory)) if str(src_directory) not in sys.path else␣
 ↪None

# Import settings.py
from settings import config

# Add configured directories from config to path
SOURCE_DIR = config("SOURCE_DIR")
sys.path.append(str(Path(SOURCE_DIR))) if str(Path(SOURCE_DIR)) not in sys.path␣
 ↪else None

QUANT_FINANCE_RESEARCH_BASE_DIR = config("QUANT_FINANCE_RESEARCH_BASE_DIR")
sys.path.append(str(Path(QUANT_FINANCE_RESEARCH_BASE_DIR))) if␣
 ↪str(Path(QUANT_FINANCE_RESEARCH_BASE_DIR)) not in sys.path else None

QUANT_FINANCE_RESEARCH_SOURCE_DIR = config("QUANT_FINANCE_RESEARCH_SOURCE_DIR")
sys.path.append(str(Path(QUANT_FINANCE_RESEARCH_SOURCE_DIR))) if␣
 ↪str(Path(QUANT_FINANCE_RESEARCH_SOURCE_DIR)) not in sys.path else None

# Add other configured directories
BASE_DIR = config("BASE_DIR")
CONTENT_DIR = config("CONTENT_DIR")
POSTS_DIR = config("POSTS_DIR")
PAGES_DIR = config("PAGES_DIR")
PUBLIC_DIR = config("PUBLIC_DIR")
SOURCE_DIR = config("SOURCE_DIR")
DATA_DIR = config("DATA_DIR")
DATA_MANUAL_DIR = config("DATA_MANUAL_DIR")

# Print system path
for i, path in enumerate(sys.path):
    print(f"{i}: {path}")
```

```
0: /usr/lib/python313.zip
1: /usr/lib/python3.13
2: /usr/lib/python3.13/lib-dynload
```

```
3:
4: /home/jared/python-virtual-envs/general_313/lib/python3.13/site-packages
5: /home/jared/Cloud_Storage/Dropbox/Websites/jaredszajkowski.github.io/src
6: /home/jared/Cloud_Storage/Dropbox/Quant_Finance_Research
7: /home/jared/Cloud_Storage/Dropbox/Quant_Finance_Research/src
```

1.3 Track Index Dependencies

[3]:
```python
# Create file to track markdown dependencies
dep_file = Path("index_dep.txt")
dep_file.write_text("")
```

[3]:
```
0
```

1.4 Python Functions

[4]:
```python
from export_track_md_deps import export_track_md_deps
```

1.5 dodo.py Functions

[5]:
```python
# Copy this <!-- INSERT_01_Import_HERE --> to index_temp.md
export_track_md_deps(dep_file=dep_file, md_filename="01_Imports.md", content=
"""
```python
########################################
## Import Libraries
########################################

import sys

## Make sure the src folder is in the path
sys.path.insert(1, "./src/")

import re
import shutil
import subprocess
import time
import yaml

from colorama import Fore, Style, init
from datetime import datetime
from os import environ, getcwd, path
from pathlib import Path
```
""")
```

```
Exported and tracked: 01_Imports.md
```

[6]:
```
# Copy this <!-- INSERT_02_Print_Green_HERE --> to index_temp.md
export_track_md_deps(dep_file=dep_file, md_filename="02_Print_Green.md",␣
 ↪content=
"""
```python
# Code from lines 29-75 referenced from the UChicago
# FINM 32900 - Full-Stack Quantitative Finance course
# Credit to Jeremy Bejarano
# https://github.com/jmbejara

## Custom reporter: Print PyDoit Text in Green
# This is helpful because some tasks write to sterr and pollute the output in
# the console. I don't want to mute this output, because this can sometimes
# cause issues when, for example, LaTeX hangs on an error and requires
# presses on the keyboard before continuing. However, I want to be able
# to easily see the task lines printed by PyDoit. I want them to stand out
# from among all the other lines printed to the console.

from doit.reporter import ConsoleReporter
from settings import config

########################################
## Slurm Configuration
########################################

try:
    in_slurm = environ["SLURM_JOB_ID"] is not None
except:
    in_slurm = False

class GreenReporter(ConsoleReporter):
    def write(self, stuff, **kwargs):
        doit_mark = stuff.split(" ")[0].ljust(2)
        task = " ".join(stuff.split(" ")[1:]).strip() + "\n"
        output = (
            Fore.GREEN
            + doit_mark
            + f" {path.basename(getcwd())}: "
            + task
            + Style.RESET_ALL
        )
        self.outstream.write(output)

if not in_slurm:
    DOIT_CONFIG = {
        "reporter": GreenReporter,
        # other config here...
```
```

```
        # "cleanforget": True, # Doit will forget about tasks that have been␣
 ↪cleaned.
        "backend": "sqlite3",
        "dep_file": "./.doit-db.sqlite",
    }
else:
    DOIT_CONFIG = {
        "backend": "sqlite3",
        "dep_file": "./.doit-db.sqlite"
    }
init(autoreset=True)
```
""")
```

```
Exported and tracked: 02_Print_Green.md
```

[7]:
```
# Copy this <!-- INSERT_03_Directory_Variables_HERE --> to index_temp.md
export_track_md_deps(dep_file=dep_file, md_filename="03_Directory_Variables.
 ↪md", content=
"""
```python
########################################
## Set directory variables
########################################

BASE_DIR = config("BASE_DIR")
CONTENT_DIR = config("CONTENT_DIR")
POSTS_DIR = config("POSTS_DIR")
PAGES_DIR = config("PAGES_DIR")
PUBLIC_DIR = config("PUBLIC_DIR")
SOURCE_DIR = config("SOURCE_DIR")
DATA_DIR = config("DATA_DIR")
DATA_MANUAL_DIR = config("DATA_MANUAL_DIR")
```
""")
```

```
Exported and tracked: 03_Directory_Variables.md
```

1.6 Complete dodo.py File

```
"""Execute with `doit` in the terminal."""

########################################
## Import Libraries
########################################

import sys
```
```

```
## Make sure the src folder is in the path
sys.path.insert(1, "./src/")

import re
import shutil
import subprocess
import time
import yaml

from colorama import Fore, Style, init
from datetime import datetime
from os import environ, getcwd, path
from pathlib import Path

# Code from lines 29-75 referenced from the UChicago
# FINM 32900 - Full-Stack Quantitative Finance course
# Credit to Jeremy Bejarano
# https://github.com/jmbejara

## Custom reporter: Print PyDoit Text in Green
# This is helpful because some tasks write to sterr and pollute the output in
# the console. I don't want to mute this output, because this can sometimes
# cause issues when, for example, LaTeX hangs on an error and requires
# presses on the keyboard before continuing. However, I want to be able
# to easily see the task lines printed by PyDoit. I want them to stand out
# from among all the other lines printed to the console.

from doit.reporter import ConsoleReporter
from settings import config

########################################
## Slurm Configuration
########################################

try:
    in_slurm = environ["SLURM_JOB_ID"] is not None
except:
    in_slurm = False

class GreenReporter(ConsoleReporter):
    def write(self, stuff, **kwargs):
        doit_mark = stuff.split(" ")[0].ljust(2)
        task = " ".join(stuff.split(" ")[1:]).strip() + "\n"
        output = (
            Fore.GREEN
            + doit_mark
            + f" {path.basename(getcwd())}: "
            + task
```

```
            + Style.RESET_ALL
        )
        self.outstream.write(output)

if not in_slurm:
    DOIT_CONFIG = {
        "reporter": GreenReporter,
        # other config here...
        # "cleanforget": True, # Doit will forget about tasks that have been cleaned.
        "backend": "sqlite3",
        "dep_file": "./.doit-db.sqlite",
    }
else:
    DOIT_CONFIG = {
        "backend": "sqlite3",
        "dep_file": "./.doit-db.sqlite"
    }
init(autoreset=True)

########################################
## Set directory variables
########################################

BASE_DIR = config("BASE_DIR")
CONTENT_DIR = config("CONTENT_DIR")
POSTS_DIR = config("POSTS_DIR")
PAGES_DIR = config("PAGES_DIR")
PUBLIC_DIR = config("PUBLIC_DIR")
SOURCE_DIR = config("SOURCE_DIR")
DATA_DIR = config("DATA_DIR")
DATA_MANUAL_DIR = config("DATA_MANUAL_DIR")

########################################
## Helper functions
########################################

def copy_file(origin_path, destination_path, mkdir=True):
    """Create a Python action for copying a file."""

    def _copy_file():
        origin = Path(origin_path)
        dest = Path(destination_path)
        if mkdir:
            dest.parent.mkdir(parents=True, exist_ok=True)
        shutil.copy2(origin, dest)

    return _copy_file
```

```
def extract_front_matter(index_path):
    """Extract front matter as a dict from a Hugo index.md file."""
    text = index_path.read_text()
    match = re.search(r"(?s)^---(.*?)---", text)
    if match:
        return yaml.safe_load(match.group(1))
    return {}

def notebook_source_hash(notebook_path):
    """Compute a SHA-256 hash of the notebook's code and markdown cells. This includes all whi
    import nbformat
    import hashlib

    with open(notebook_path, "r", encoding="utf-8") as f:
        nb = nbformat.read(f, as_version=4)

    relevant_cells = [
        cell["source"]
        for cell in nb.cells
        if cell.cell_type in {"code", "markdown"}
    ]
    full_content = "\n".join(relevant_cells)
    return hashlib.sha256(full_content.encode("utf-8")).hexdigest()

def clean_pdf_export_pngs(subdir, notebook_name):
    """Remove .png files created by nbconvert during PDF export."""
    pattern = f"{notebook_name}_*_*.png"
    deleted = False
    for file in subdir.glob(pattern):
        print(f"  Removing nbconvert temp image: {file}")
        file.unlink()
        deleted = True
    if not deleted:
        print(f"  No temp PNGs to remove for {notebook_name}")

########################################
## PyDoit tasks
########################################

def task_config():
    """Create empty directories for content, page, post, and public if they don't exist"""
    return {
        "actions": ["ipython ./src/settings.py"],
        "file_dep": ["./src/settings.py"],
        "targets": [CONTENT_DIR, PAGES_DIR, POSTS_DIR, PUBLIC_DIR],
        "verbosity": 2,
        "clean": [],  # Don't clean these files by default.
    }
```

```
def task_list_posts_subdirs():
    """Create a list of the subdirectories of the posts directory"""
    return {
        "actions": ["python ./src/list_posts_subdirs.py"],
        "file_dep": ["./src/settings.py"],
        # "targets": [POSTS_DIR],
        "verbosity": 2,
        "clean": [],  # Don't clean these files by default.
    }

def task_run_post_notebooks():
    """Execute notebooks that match their subdirectory names and only when code or markdown co
    for subdir in POSTS_DIR.iterdir():
        if not subdir.is_dir():
            continue

        notebook_path = subdir / f"{subdir.name}.ipynb"
        if not notebook_path.exists():
            continue  #  Skip subdirs with no matching notebook

        hash_file = subdir / f"{subdir.name}.last_source_hash"
        log_file = subdir / f"{subdir.name}.log"

        def source_has_changed(path=notebook_path, hash_path=hash_file, log_path=log_file):
            current_hash = notebook_source_hash(path)
            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

            if hash_path.exists():
                old_hash = hash_path.read_text().strip()
                if current_hash != old_hash:
                    print(f"  Change detected in {path.name}")
                    return False  # needs re-run

                #  No change → log as skipped
                with log_path.open("a") as log:
                    log.write(f"[{timestamp}]  Skipped (no changes): {path.name}\n")
                print(f"  No change in hash for {path.name}")
                return True

            #  No previous hash → must run
            print(f"  No previous hash found for {path.name}")
            return False

        def run_and_log(path=notebook_path, hash_path=hash_file, log_path=log_file):
            start_time = time.time()
            subprocess.run([
                "jupyter", "nbconvert",
```

```
                "--execute",
                "--to", "notebook",
                "--inplace",
                "--log-level=ERROR",
                str(path)
            ], check=True)
            elapsed = round(time.time() - start_time, 2)

            new_hash = notebook_source_hash(path)
            hash_path.write_text(new_hash)
            print(f"  Saved new hash for {path.name}")

            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            log_msg = f"[{timestamp}]   Executed {path.name} in {elapsed}s\n"
            with log_path.open("a") as f:
                f.write(log_msg)

            print(log_msg.strip())

        yield {
            "name": subdir.name,
            "actions": [run_and_log],
            "file_dep": [notebook_path],
            "uptodate": [source_has_changed],
            "verbosity": 2,
        }

def task_export_post_notebooks():
    """Export executed notebooks to HTML and PDF, and clean temp PNGs"""
    for subdir in POSTS_DIR.iterdir():
        if not subdir.is_dir():
            continue

        notebook_name = subdir.name
        notebook_path = subdir / f"{notebook_name}.ipynb"
        html_output = subdir / f"{notebook_name}.html"
        pdf_output = subdir / f"{notebook_name}.pdf"

        if not notebook_path.exists():
            continue

        yield {
            "name": notebook_name,
            "actions": [
                f"jupyter nbconvert --to=html --log-level=WARN --output={html_output} {noteboo
                f"jupyter nbconvert --to=pdf --log-level=WARN --output={pdf_output} {notebook_
                (clean_pdf_export_pngs, [subdir, notebook_name])
            ],
```

```
            "file_dep": [notebook_path],
            "targets": [html_output, pdf_output],
            "verbosity": 2,
            "clean": [],  # Don't clean these files by default.
        }

def task_build_post_indices():
    """Run build_index.py in each post subdirectory to generate index.md"""
    script_path = SOURCE_DIR / "build_index.py"

    for subdir in POSTS_DIR.iterdir():
        if subdir.is_dir() and (subdir / "index_temp.md").exists():
            def run_script(subdir=subdir):
                subprocess.run(
                    ["python", str(script_path)],
                    cwd=subdir,
                    check=True
                )

            yield {
                "name": subdir.name,
                "actions": [run_script],
                "file_dep": [
                    subdir / "index_temp.md",
                    subdir / "index_dep.txt",
                    script_path,
                ],
                "targets": [subdir / "index.md"],
                "verbosity": 2,
                "clean": [],  # Don't clean these files by default.
            }

def task_clean_public():
    """Remove the Hugo public directory before rebuilding the site."""
    def remove_public():
        if PUBLIC_DIR.exists():
            shutil.rmtree(PUBLIC_DIR)
            print(f" Deleted {PUBLIC_DIR}")
        else:
            print(f"  {PUBLIC_DIR} does not exist, nothing to delete.")
    return {
        "actions": [remove_public],
        "verbosity": 2,
        "clean": [],  # Don't clean these files by default.
    }

def task_build_site():
    """Build the Hugo static site"""
```

```
    return {
        "actions": ["hugo"],
        "task_dep": ["clean_public"],
        "verbosity": 2,
        "clean": [],  # Don't clean these files by default.
    }

def task_copy_notebook_exports():
    """Copy notebook HTML exports into the correct Hugo public/ date-based folders"""
    for subdir in POSTS_DIR.iterdir():
        if subdir.is_dir():
            html_file = subdir / f"{subdir.name}.html"
            index_md = subdir / "index.md"

            if not html_file.exists() or not index_md.exists():
                continue

            # Extract slug and date from front matter
            front_matter = extract_front_matter(index_md)
            slug = front_matter.get("slug", subdir.name)
            date_str = front_matter.get("date")
            if not date_str:
                continue

            # Format path like: public/YYYY/MM/DD/slug/
            date_obj = datetime.fromisoformat(date_str)
            public_path = PUBLIC_DIR / f"{date_obj:%Y/%m/%d}" / slug
            target_path = public_path / f"{slug}.html"

            def copy_html(src=html_file, dest=target_path):
                dest.parent.mkdir(parents=True, exist_ok=True)
                shutil.copy2(src, dest)
                print(f"  Copied {src} → {dest}")

            yield {
                "name": subdir.name,
                "actions": [copy_html],
                "file_dep": [html_file, index_md],
                "targets": [target_path],
                "task_dep": ["build_site"],
                "verbosity": 2,
                "clean": [],  # Don't clean these files by default.
            }

def task_create_schwab_callback():
    """Create a Schwab callback URL by creating /public/schwab_callback/index.html and placing
    def create_callback():
        callback_path = PUBLIC_DIR / "schwab_callback" / "index.html"
```

```
        callback_path.parent.mkdir(parents=True, exist_ok=True)
        html = """<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Schwab OAuth Code</title>
    <script>
        const params = new URLSearchParams(window.location.search);
        const code = params.get("code");
        document.write("<h1>Authorization Code:</h1><p>" + code + "</p>");
    </script>
</head>
<body></body>
</html>"""
        with open(callback_path, "w") as f:
            f.write(html)
        print(f"  Created Schwab callback page at {callback_path}")

    return {
        "actions": [create_callback],
        "task_dep": ["copy_notebook_exports", "clean_public"],
        "verbosity": 2,
        "clean": [],  # Don't clean these files by default.
    }

def task_deploy_site():
    """Prompt for a commit message and push to GitHub"""
    def commit_and_push():
        message = input("What is the commit message? ")
        if not message.strip():
            print("  Commit message cannot be empty.")
            return 1  # signal failure
        import subprocess
        subprocess.run(["git", "add", "."], check=True)
        subprocess.run(["git", "commit", "-am", message], check=True)
        subprocess.run(["git", "push"], check=True)
        print("  Pushed to GitHub.")

    return {
        "actions": [commit_and_push],
        "task_dep": ["create_schwab_callback"],
        "verbosity": 2,
        "clean": [],  # Don't clean these files by default.
    }

# def task_build_all():
#     return {
#         "actions": None,
```

```
#         "task_dep": [
#             "run_post_notebooks",
#             "export_post_notebooks",
#             "build_post_indices",
#             "clean_public",
#             "build_site",
#             "copy_notebook_exports",
#             "create_schwab_callback",
#             "deploy_site",
#         ]
#     }
```