# Baroque TOC

# Automating Exhibition Catalogue Creation

—A Guide

by ADA Team Editorial

v1.0

Published by: ADA Semantic Publishing Pipeline

Last updated: 2023-05-04

Created: 2023-04-28

Language: English (United States)

Created by: Editorial Team

# Contents

# About the prototype

## Publication type: Use case - An exhibition catalogue

1. We are creating a *demonstration prototype*: **An exhibition catalogue about a baroque painting collection.**
2. Objectives:
    1. Write an exhibition catalogue essay using AI tools
    2. Review the 'catalogue essay and AI tools' as open peer review
    3. Create the parts of the the catalogue:
        1. Cover
        2. Colophon
        3. Essay
        4. Collection
3. What is the collection?
    1. The catalogue uses **part** of a Wikidata based collection of Bavarian collections of Baroque paintings. See: 17C Bavarian painting.
    2. We focus on the Baroque period: Bavarian Collections, 1590-1750 query link
    3. We make a small collection of paintings - 9 in this case.
4. How are we using computational publishing and what is the prototype experiment?
    1. Creating a publication from different **distributed** (federated) remote sources using linked open data.
    2. Showing how asyncronous work can be carried out by team working on a single publication - this is the power of the **TOC** part! Which in more advanced domains becomes **package management**.

# Learning points

Workflow activities that will be covered to create the exhibition catalogue:

1. Real-time collaborative editing,
2. Creating a Wikidata query of a collection,
3. Displaying a painting catalogue sample collection from Wikidata LOD query for a multi-format publication.
4. Editing a Jupyter Notebook in MyBinder,
5. Embedding media objects: Video - TIB AV Portal, and; Semantic Kompakkt,
6. Using GitHub
7. Accessing API content for colophon
8. Editing Wikidata collection query in Juypter Notebooks
9. Asycrononous collective working and making a publication from multiple remote Linked Open Data (LOD) sources, and
10. Rendering a multi-format publication with CSS styling.

# Software (open-source)

Over 2023/24 the computational components will be added to the ***ADA Semantic Publishing Pipeline*** as well as introducing **Vivliostyle Create Book** markdown renderer and swapping to **Jupyter Book** computational book platform away from Quarto – https://github.com/NFDI4Culture/ada

- Wikidata – https://www.wikidata.org/
- Jupyter Notebooks – https://jupyter.org/
- Jupyter Book – https://jupyterbook.org/
- Quarto – https://quarto.org/
- Semantic Kompakkt – https://semantic-kompakkt.de/
- TIB AV Portal – https://av.tib.eu/
- HedgeDoc – https://HedgeDoc.org/
- Thoth – https://thoth.pub/
- Vivliostyle – https://vivliostyle.org/
    - Create Book – Markdown renderer
- Wikibase – https://wikiba.se/
- Zenodo - https://zenodo.org/
- NextCloud - Tetx editor and Markdown editor - Text : https://github.com/nextcloud/text Markdown: https://apps.nextcloud.com/apps/files_markdown

## AI Software

To be confirmed

https://openai.com/blog/chatgpt

https://www.perplexity.ai/

# Activity: Editing a Jupyter Notebooks and accessing video

**Objective:** Running and editing Juypter Notebooks in MyBinder and retrieving video and 3D models as embeds.

**External LOD and media used:** TIB AV Portal, and Semantic Kompakkt

**Notes:** Jupyter Notebooks editing in MyBinder

- Run a Jupyter Notebook in MyBinder
- Edit a Jupyter Notebook
- Render a Jupyter Notebooks

**Links:**

- Sample Jupyter Notebook: Video and 3D Notebook embeds
- TIB AV Portal: https://av.tib.eu/
- Semantic Kompakkt demo site: https://kompakkt.wbworkshop.tibwiki.io/explore
- View a model, copy the iframe embed from the folder icon, top right. In the Notebook paste in the complete iframe cover replacing the existing iframe: <iframe name="Doric Column" src="https://kompakkt.wbworkshop.tibwiki.io/viewer/?entity=63e8c22910e4f555d1f656ca&mode=open" allowfullscreen loading="lazy" ></iframe>

## Steps

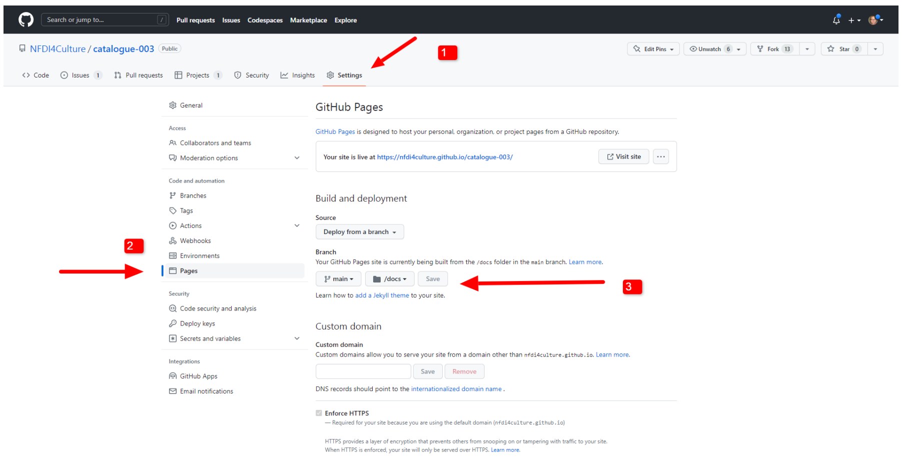
1. Open Notebook in the browser using MyBinder - Video and 3D Notebook embeds - click the 'launch binder' button to run the Notebook in MyBinder.
2. Add new videos and 3D models to the Notebook from TIB AV Portal and Semantic Kompakkt.
    1. Open a second browser tab and load TIB AV Portal
    2. Choose a video and copy across the video ID from the URL https://av.tib.eu/media/60729
    3. Paste the video ID into the video iframe field and run the cell to render
    4. Open Semantic Kompakkt demo site: https://kompakkt.wbworkshop.tibwiki.io/explore
    5. View a model, copy the iframe embed from the folder icon, top right. In the Notebook paste in the complete iframe cover replacing the existing iframe: <iframe name="Doric Column" src="https://kompakkt.wbworkshop.tibwiki.io/viewer/?entity=63e8c22910e4f555d1f656ca&mode=open" allowfullscreen loading="lazy" ></iframe>
3. Run the Notebook
4. 3D view size, we can make the initial view bigger, add: <iframe width="1200" height="630"
5. Download Notebook
6. Render some videos and 3D models in the Quarto book. Pass along **video id codes** and **3d models** using a hedge doc and chat to the Quarto render. The rendering and final display will take less than 10 minutes (hopefully): a. The code needs to be added to the main repo; b. Rendered locally; c. Uploaded to GitHub; d. Time for GitHub Pages to finish loading.
    1. Code: https://github.com/SimonXIX/Experimental_Books_workshop/blob/main/paintings.ipynb
    2. Rendering: https://simonxix.github.io/Experimental_Books_workshop/paintings.html

# Activity: GitHub on boarding

**Objective:** On boarding and familiarisation with using GitHub for publishing and asynchronous working.

Publication repository: https://github.com/NFDI4Culture/catalogue-003

- Creating an account
- Joining an organisation
- Forking a repository
- Cloning a repository - use GitHub Desktop, Visual Studio Code, or any other tool to copy the repository to your local machine
- **IMPORTANT!:** Turning on Github Pages.
    - First go to the Settings tab in your repository; second in the left menu go down to Pages; third, secect main, docs - and save. In a few minutes this will torn on your Github Pages website. Congrats!



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at https://nfdi4culture.github.io/catalogue-003/

Build and deployment

Source: Deploy from a branch

Branch

Your GitHub Pages site is currently being built from the `/docs` folder in the `main` branch. Learn more.

Learn how to add a Jekyll theme to your site.

Custom domain

Custom domain

Custom domains allow you to serve your site from a domain other than `nfdi4culture.github.io`. Learn more.

DNS records should point to the internationalized domain name .

Enforce HTTPS

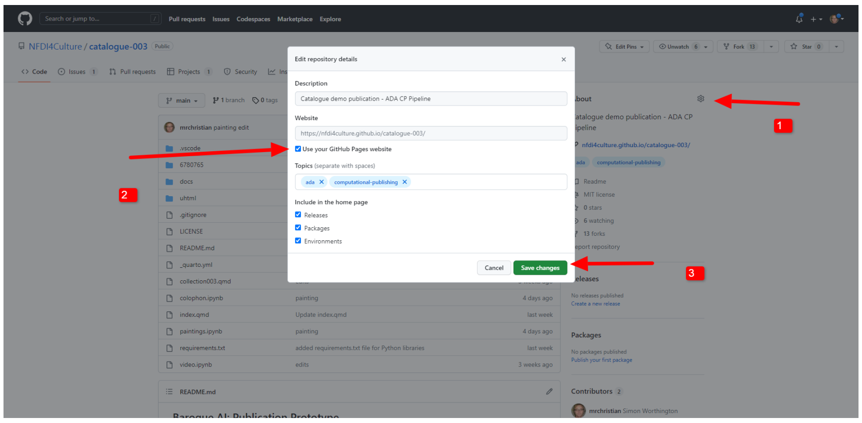
— Required for your site because you are using the default domain (`nfdi4culture.github.io`)

HTTPS provides a layer of encryption that prevents others from snooping on or tampering with traffic to your site. When HTTPS is enforced, your site will only be served over HTTPS. Learn more.

- **Figure 1**: Turn on GitHub Pages

- The last steup is to add the GitHub Pages URL to the front end information panel of your repository.

- Navigate back ton the Code view of your repo. Top your you can add your GitHub Pages URL to the About information of your repository. Open the About area by clicking on the cog icon. The in the dialog window click the use Pages address, and save.


Edit repository details

Description: Catalogue demo publication - ADA CP Pipeline

Website: https://nfdi4culture.github.io/catalogue-003/

☑ Use your GitHub Pages website

Topics (separate with spaces): ada, computational-publishing

Include in the home page

☑ Releases
☑ Packages
☑ Environments

- **Figure 2**: Add Pages URL to repo frontend

- Enabling a local editor: Visual Studio Code
- Attribution and citation

# Quarto Install

Quarto help docs: https://quarto.org/

## Options

For general purposes use the manual install. The Docker install is for when you are running multiple environments on your computer or carrying out long term development.

- Manual install
- Docker install
- Visual Code Studio install (not covered here)

## Manual install

https://quarto.org/docs/get-started/

Note: They miss out that you need Python installed and Jupyter Lab and a working terminal to install them

https://www.python.org/downloads/

https://jupyter.org/install

Also install Panda: py -m pip install panda

As well as knowing what the Python prompt and escape looks like: https://stackoverflow.com/questions/41524734/how-to-exit-python-script-in-command-prompt

If on Windows pip or python wont run. Check the solution here to add the python path for the Terminal April 23.

## Docker install

### Docker Compose

Text source: Readme - Thanks to Simon Bowie

---

https://github.com/NFDI4Culture/cp4c

This repository also contains Docker Compose and Dockerfiles for running the various applications in Docker containers.

Run docker-compose up -d --build to start the containers and docker-compose down to stop the containers.

The jupyterlab container runs a stand-alone version of JupyterLab on http://localhost:8888. This can be used to edit any Jupyter Notebook files in the repository. The JupyterLab instance runs with the password 'jupyterlab'.

The nginx container runs Nginx webserver and displays the static site that Quarto renders. This runs at http://localhost:1337.

The quarto container starts a Ubuntu 22.04 container, installs various things like Python, downloads Quarto and installs it, and then adds Python modules like jupyter, matplotlib, and panda. It then runs in the background so Quarto can be called on to render the qmd and ipynb files into the site/book like so:

docker exec -it quarto quarto render

There's a known issue with Quarto running in the Docker container in macOS due to the amd64 emulation of Docker Desktop for arm64 MacOS. See discussion at quarto-dev/quarto-cli#3308. This shouldn't occur in any other environment running Docker.

### Visual Code Studio - Quarto install

This is not covered at present as there are conflicts with other approaches.

## Troubleshooting

Raise an issue in the GitHub project if you need support.

### Quarto commands

quarto render

quarto preview

quarto check

# Quarto Rendering

Quarto help docs: https://quarto.org/

Using quarto to render as multi-format.

Use the command line to run Quarto: Powershell, GitBash, Cygwin, OSX shell, or terminal in Visual Studio Code ('VSC), etc.

Execute the Quarto commands from the top level of your publication repository.

## Steps

- **IMPORTANT!** Run and save Jupyter Notebooks, install the **requirements.txt** file before running Notebooks. SAVE all files.
- Work locally with **quarto preview** - this will launch a browser window to preview your publication.
- Render: Use command **quarto render.**
- Upload to GitHub when ready using commit and push - Use GitHub Desktop, or from V'SC, etc.

## Troubleshooting

Problems encountered so far:

1. Cover image: file needs to be local for epub rendering
2. Python path on Windows: If on Windows pip or python wont run. Check the solution here to add the python path for the Terminal April 23.
3. CMD.EXE - Quarto wont run. See https://www.windows-faq.de/2017/02/27/unc-pfade-in-der-eingabeaufforderung-benutzen/ (no fix as yet - April 2023.
4. Moving a repository locally: From Stackoverflow. If you are using GitHub Desktop, then just do the following steps:
    1. Close GitHub Desktop and all other applications with open files to your current directory path.
    2. Move the whole directory as mentioned above to the new directory location. **(NB: The fdirectory has to be completely moved).**

3. Open GitHub Desktop and click on the blue (!) "repository not found" icon. Then a dialog will open and you will see a "Locate..." button which will open a popup allowing you to direct its path to a new location.

# Editing Quarto

Visual Studio Code is one option as an editor, but you can use any editor suite that you like.

## Install Visual Studio Code (VSC)

https://code.visualstudio.com/

### Editing

Load the whole repository folder into your editor.

The key file to edit in Quart is **_quarto.yml.** This file contains the main configurations for your publication.

If you are working on a fork the first thing you need to do is edit the repository address on line 19 - this will point the GitHub icon in your publication to your own GitHub repo.

> *repo-url: https://github.com/NFDI4Culture/catalogue-003*

To add new sections to your publication jast add file names to the chapter list after line 12

The other settings can we read about on the Quarto support pages, for example Book Structure.

If you are using VSC you can run Notebooks as well as use the Terminal to run Quarto commands, and commit to GitHub.

# Jupyter Notebooks: Setup, Editing, and Saving

General instructions for using Jupyter Notebooks.

Installation of Python and Jupyter Notebooks is covered in Quarto install section.

The default editor we use is Visual Code Studio, but you can use other Notebook editors.

## Steps

First run: **requirements.txt** file for Python libraries. See: https://note.nkmk.me/en/python-pip-install-requirements/

Use:

```
# install with 'pip install -r requirements.txt'
```

Then you can run, edit, and save Notebooks.

# Activity: Create a Wikidata query

**Objective:** User builds a Wikidata query. See example query: paintings, Bavarian Collections, 1590 - 1750 - query link

**External LOD and media used:** Wikidata LOD, and Wiki Commons, Web Gallery of Art (images) - https://www.wga.hu/

**Notes:** Wikidata Query (help)

- Allows for non-expert query building with plain language
- View query as plain language and as code
- Experience of building a query
- Contact with some basic building blocks of Wikidata
- View and export SPARQL query

## Steps

1. Go to https://query.wikidata.org/
2. Build a query around the 17C Bavarian painting collection to replicate the catalogue selection to be used in Activity B. Example:
    1. Code Repo: Current publication link
    2. Rendering: Current publication link
    3. Example: Paintings; in collection; Bavarian Collections; 1600 - 1700 - query link
3. ***Step-by-step instructions*** to replicate parts of this query link base on this collection 17C Bavarian painting: **Note: You will use the lefthand 'Query helper' GUI, where you can type in names of items. Sometimes you need to enter a term twice to get the correct item to appear.** Items names are in bold below with their corresponding identifiers.
    1. Go to https://query.wikidata.org/
    2. Enable split view with ***i info*** button top left.

3. Filter: **instance of** P31, **painting** Q3305213 - wdt:P31 wd:Q3305213.
4. Filter: **collection** P195, **Bavarian State Painting Collection** Q812285 - wdt:P195 wd:Q812285.
5. Play button - bottom left - renders query below
6. Show: **creator** P170; **image** P18; **copyright status** P6216; **inception** P571.
7. Play button - bottom left - renders query below
8. Image grid view :-)
9. Limit
10. Dates from to 1590-1750 (code only) BIND(YEAR(?inception) AS ?inceptionyear) FILTER((1590 <= ?inceptionyear) && (?inceptionyear < 1750 ))
11. Link query: https://w.wiki/6MGX - results: https://w.wiki/6MGY

4. Participants can change the selection criteria around the available criteria: artists, dates, etc., as in collection 17C Bavarian painting
5. Completion: Paste your query link into the HedgeDoc link provided. https://demo.HedgeDoc.org/s/4gr9JvUS7 - END of activity.

# Activity: Making SPARQL Queries in Wikidata of Collections

Keywords: Collections, Wikidata, SPARQL

If you are not familiar with creating a Wikidata SPARQL query then see the section on how to create a query: **Activity: Create a Wikidata query**

## The activity goal

**Create a sample SPARQL query of a cultural collection of a museum and save a link to the query on your Wikiversity user page.**

Note: ***Two collections are needed*** as the painting collection is easier to add to Jupyter Notebooks as you edit existing data. The second general collection is used to explore more options - assistence will be needed to help move the second query to the Jupyter Notebooks as they can be complex.

1. **First one of a painting collection, and then,**
2. **A second one - of any type of media or artifact from a collection: books, sculpture, photography, etc.**

Here is an example: Bavarian Collections, 1590 - 1750 SW March 2023

If you need support the GitHub Project (task tracker) for the class is here - titled: *Prototype catalogue - ADA CP pipeline*.

Later in the workflow the query will be added to a Jupyter Notebook for rendering in Quarto.

**Create the two queries and add the query links to Wikiversity.**

## Query 1: A painting collection

The Painting Notebook is simpler to move to Jupyter Notebooks as the number of edits needed are fewer.

1. Use and copy the existing painting collection and modify the setting Bavarian Collections, 1590 - 1750.
2. The exercise is to change three parts: the painting collection, date perameters, and number of items.

    1. **Collection:** Line 9 - *?item wdt:P195 wd:Q812285*: Add in the identifier of a different collection - see list here Here is a sample list of painting collections found on Wikidata, Wikidata:WikiProject sum of all paintings/Collection. As an example here the the Frick Museum - https://www.wikidata.org/wiki/Q682827 has a unique identifier of unique Q682827.
    2. **Date:** Line 37 - *FILTER((1590 <= ?inceptionyear) && (?inceptionyear < 1750 ))*: here change the from and to dates.
    3. **Number of items:** Line 40 - *LIMIT 9*: Change the number of items and run preview.

    Now copy the query URL and save the link to your Wikiversiyt page. To copy the link you need to have previewed the query and bottom right is a short link copy button (if this doesnt work - copy the URL from your browsers address bar). Later we will add the new query to the Notebook painting.ipny.

## Query 2: Make your own collection query

Note: We want to have images in our query, using the image grid view allows you to preview images.

### Resources for finding collections: Data models and information

The links will help you find collections and see what types of artwork or media are listed on Wikidata.

**The help and examples in the query service are useful for seeing examples and modifying them https://query.wikidata.org/.**

The examples section has a RUN IT link under each examples where the query will load for you. Note to check the image grid view.

As example here is one of scultures by Max Bill. You can use this and change the artists name, for example to Donatello.

|  |  |
|---|---|
| *Museums* | *Film* |
| WikiProject sum of all paintings | Art genres |
| Books | GLAM WikiProjects |

## Steps for making your own query

1. Start with a blank query https://query.wikidata.org/
2. Use the **Examples** button top left and then edit the example, or use DuckduckGo or Google to search for queries to edit.
3. Save your query link on your Wikiversity user homepage.

---

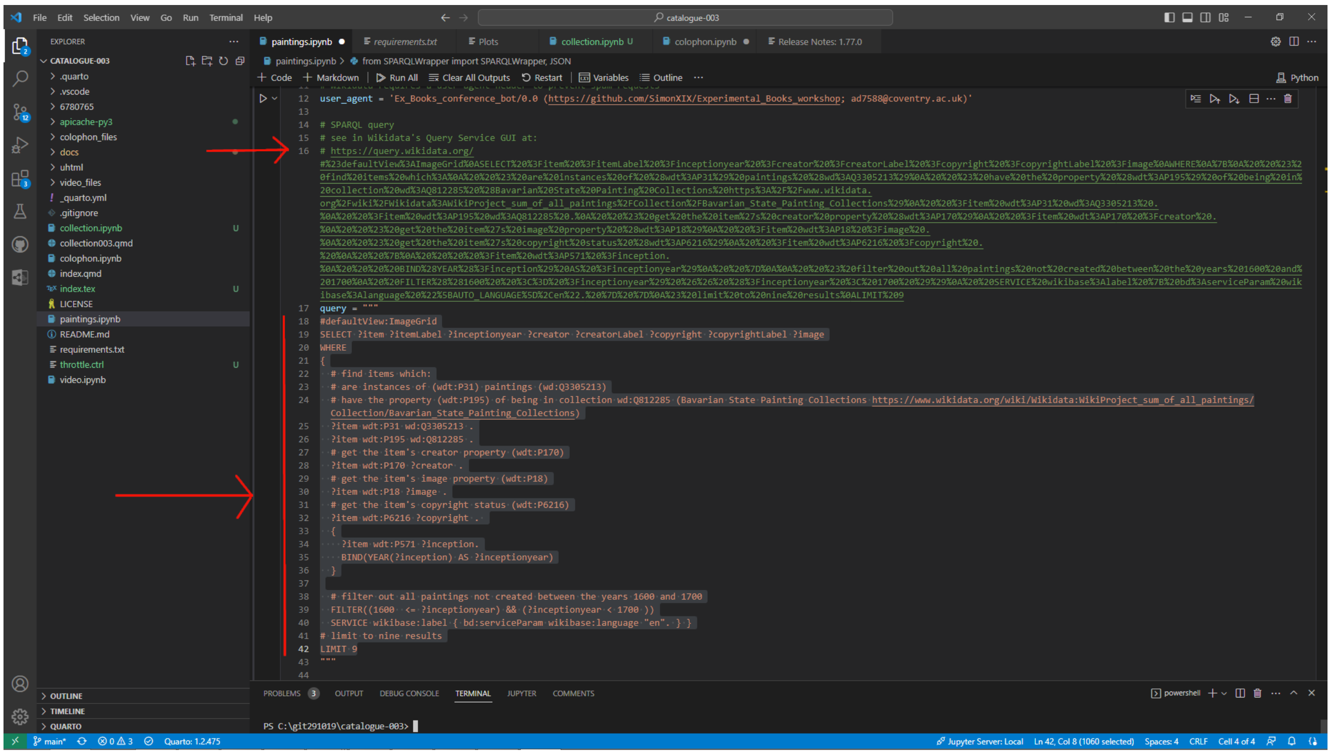
In the next section ***'Transfering a Wikidata SPARQL query to a Juypter Notebook'*** you will see how to move your queries to Jupyter Notebooks.

# Transfering a Wikidata SPARQL query to a Juypter Notebook

- Query 1: A painting collection - can already be rendered in the Notebook and by Quarto.
- Query 2: Yourn own collection query - you will need assistance in the class to render this as the outputted SPARQL metadata results needs to be processed by some Python code and this quickly becomes complicated.

## Query 1: A painting collection

For this collection you simply edit the existing Notebook for the painting collection **'paintings.ipynb'** rename as **paintings_1xx.ipynb** (with xx as your initails) - transfering across the new values that you have already created. See the section 'Making SPARQL Queries in Wikidata of Collections'.



```
user_agent = 'Ex_Books_conference_bot/0.0 (https://github.com/SimonXIX/Experimental_Books_workshop; ad7588@coventry.ac.uk)'

# SPARQL query
# see in Wikidata's Query Service GUI at:
# https://query.wikidata.org/
#%23defaultView%3AImageGrid%0ASELECT%20%3Fitem%20%3FitemLabel%20%3Finceptionyear%20%3Fcreator%20%3FcreatorLabel%20%3Fcopyright%20%3FcopyrightLabel%20%3Fimage%0AWHERE%0A%7B%0A%20%20%23%2
0find%20items%20which%3A%0A%20%20%23%20are%20instances%20of%20%28wdt%3AP31%29%20paintings%20%28wd%3AQ3305213%29%0A%20%20%23%20have%20the%20property%20%28wdt%3AP195%29%20of%20being%20in%
20collection%20wd%3AQ812285%20%28Bavarian%20State%20Painting%20Collections%20https%3A%2F%2Fwww.wikidata.
org%2Fwiki%2FWikidata%3AWikiProject_sum_of_all_paintings%2FCollection%2FBavarian_State_Painting_Collections%29%0A%20%20%3Fitem%20wdt%3AP31%20wd%3AQ3305213%20.
%0A%20%20%3Fitem%20wdt%3AP195%20wd%3AQ812285%20.%0A%20%20%23%20get%20the%20item%27s%20creator%20property%20%28wdt%3AP170%29%0A%20%20%3Fitem%20wdt%3AP170%20%3Fcreator%20.
%0A%20%20%23%20get%20the%20item%27s%20image%20property%20%28wdt%3AP18%29%0A%20%20%3Fitem%20wdt%3AP18%20%3Fimage%20.
%0A%20%20%23%20get%20the%20item%27s%20copyright%20status%20%28wdt%3AP6216%29%0A%20%20%3Fitem%20wdt%3AP6216%20%3Fcopyright%20.
%20%0A%20%20%7B%0A%20%20%20%20%3Fitem%20wdt%3AP571%20%3Finception.
%0A%20%20%20%20BIND%28YEAR%28%3Finception%29%20AS%20%3Finceptionyear%29%0A%20%20%7D%0A%0A%20%20%23%20filter%20out%20all%20paintings%20not%20created%20between%20the%20years%201600%20and%
201700%0A%20%20FILTER%28%281600%20%20%3C%3D%20%3Finceptionyear%29%20%26%26%20%28%3Finceptionyear%20%3C%201700%20%29%29%0A%20%20SERVICE%20wikibase%3Alabel%20%7B%20bd%3AserviceParam%20wik
ibase%3Alanguage%20%22%5BAUTO_LANGUAGE%5D%2Cen%22.%20%7D%20%7D%0A%23%20limit%20to%20nine%20results%0ALIMIT%209
query = """
#defaultView:ImageGrid
SELECT ?item ?itemLabel ?inceptionyear ?creator ?creatorLabel ?copyright ?copyrightLabel ?image
WHERE
{
  # find items which:
  # are instances of (wdt:P31) paintings (wd:Q3305213)
  # have the property (wdt:P195) of being in collection wd:Q812285 (Bavarian State Painting Collections https://www.wikidata.org/wiki/Wikidata:WikiProject_sum_of_all_paintings/
  Collection/Bavarian_State_Painting_Collections)
  ?item wdt:P31 wd:Q3305213 .
  ?item wdt:P195 wd:Q812285 .
  # get the item's creator property (wdt:P170)
  ?item wdt:P170 ?creator .
  # get the item's image property (wdt:P18)
  ?item wdt:P18 ?image .
  # get the item's copyright status (wdt:P6216)
  ?item wdt:P6216 ?copyright .
  {
    ?item wdt:P571 ?inception.
    BIND(YEAR(?inception) AS ?inceptionyear)
  }

  # filter out all paintings not created between the years 1600 and 1700
  FILTER((1600  <= ?inceptionyear) && (?inceptionyear < 1700 ))
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en". } }
# limit to nine results
LIMIT 9
"""
```

```
PS C:\git291019\catalogue-003>
```

**Figure 3**: Where to add your Wikidata SPARQL URL link and paste in the full query.

These are the values being changed: **Collection, Date range, and Number of items.**

1. All you need to do is paste in two items into the main cell of the Notebook:
    1. The SPARQL URL - see the green text indicated in Figure 1
    2. Paste in the complete body of the SPARQL query - see the orange text in Figure 1. Note all of your query is pasted between the four apostrohies top and bottom: """ your query """"

Then run the Notebook and save the Notebook file when you are done and any other files edited.

Note: Change the notebook name in the _quarto.yml TOC so that your new file is included in the publication. Change paintings.ipynb to paintings_1xx.ipynb.

Note: Ensure requirements.txt has been added and run.

To render the output in Quarto, run the Quarto commands in your terminal - 'quarto preview' and/or 'quarto render'.

When your finished you can upload the results to your GitHub repository.

## Query 2: Your custom collection query

Because SPARQL queries are varied and complex the process of moving the query into a Jupyter Notebook is not simple and involved knowledge of SPARQL and Python. The challenge is that the SPARQL metadata output has to be parsed by Python for presentation in the publication and to do this custom code needs to be written.

**To help with this obstacle a support service is in place to consult and add the Python code to your Notebooks.**

When you have completed the steps below and have a Notebook ready raise a ticket in the GitHub Project and assign to support and we will review the Notebook.

See all class members collection notebooks here. This is a test zone where support will edit copies of your notebooks to show how to make them woirk: https://github.com/NFDI4Culture/sandbox-notebook-rendering

### Steps

This will prepare the Notebook to output the SPARQL results only. The additional rendering will need to be added by support.

Create a new Notebook in your Quarto publication 'collection-name.ipynb'.

Back in the custom collection query at the bottom right there is a <code> button - click on the button and it will launch a preview window. Top right click on the Python tab, this will present the Query as Python code. Now copy and paste the code into your Notebook.

Next, we want to copy the URL of your query to then also past this into your notebook. This will allow users to review your orginal query. The URL is found on the bottom right of the query page, as copy short link. If this does not work just copy the long address from your browser address bar.

Paste your query link into the Notebook as a comment, which means haveing a # at the start of the line. Like so, underneath the lines shown here:

> *import sys*
>
> *from SPARQLWrapper import SPARQLWrapper, JSON*
>
> *# query link - https://w.wiki/6YJi based on https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples#Paintings_by_Gustav_Klimt*

To finish your Notebook editing insert a Markdown field above your code cell as add a desciption, e.g.,

> *Custom Collection by Simon Worthington (April 23): Paintings by Lucas Cranach the Elder https://www.wikidata.org/wiki/Q191748.*
>
> *The collection Notebook only contains the SPARQL query and needs additional Python adding to parse the metadata output.*

Save and run your Notebook. The result will be the SPARQL unformatted output.

Add your Notebook to the Quarto TOC by adding the file name of your collection to the TOC in the file **_quarto.yml**, under the chapter header on line 12. Save and render the quart publication to see the results.

To complete this part raise a ticket in the GitHub project assigning to support.

This concludes this part of the process.

# Publishing Tasks

These are the small tasks needed for getting ready to publish. There will be an additional Flight Check round after this before completeing publishing.

1. Add a cover image
2. Update Imprint (colophon)
3. Add Essay
4. Change Style
5. Update the readme information
6. Tidy up the TOC

- **Add a cover image: Add image - instructions from Quarto help.**
    - You need to make a cover image and add link to the _quarto.yml configuration file. The image size can be any size. Recommend to use 2,560 × 1,600 pixels which is Amazon Marketing Cover Image size.

Upload your image to the top level of repository.

Then link in the file _quarto.yml at the link cover-image: as for example

Cover-image: cover.jpg

See example  _quarto.yml cover link. File can be in repo or online.

- **Update Imprint (colophon): file name - colophon.ipynb**

The colophon is made of two parts. The first part that you will edit using instructions below. And the second part which automatically comes for the Thoth.pub book metadata system.

- To get a DOI you need to register one at Zenodo. Make a Zenodo deposit for a book: Make a pre-release and get a DOI. You only need the minimal information to start with and the fields can be changed later. See Zenodo help - search for' *Reserve DOI* '.
- Add a Markdown cell after the Notebook metadata cell. Here is an example you can copy. Add and fill these fields:
    - Fork title

        - Author
        - ORCID
        - Date
        - DOI
        - Repository URL
    - **Add Essay**
        - Paste and edit Markdown here collection003.qmd. The essay is only for illustration purposes. Feel free to use Wikipedia or some AI content. Any content must be cited and be open licensed.
    - **Change Style**
        - See a list of styles https://quarto.org/docs/output-formats/html-themes.html e.g., ***journal***. To change the style simply add the style to the _quarto.yml (in this example see line 36) file and re-render the publication.
    - **Update the readme information**
        - See files README.md and index.qmd. Add minimum publication information, which can be a copy of your colophon. See the example file.
    - **Tidy up the TOC**
        - Remove other Notebooks from the chapter list in _quarto and add any other Notebooks or files to the chapter list. Remember to render and push to complete the publishing.
            - # put a hash in front of chapters to omit
            - Mynewchapter.qmd or .ipynb

## Parts of the book ↣ Files to modify or add

**Quick reference file look-up.**

Note: **_quarto.yml** is where all book configurations are done for Quarto.

**Cover**

- **_quarto.yml** at line cover-image:
    - either edit web address
    - or add cover file to add in [directory]

**Colophon**

- **colophon.ipynb**

    - (1st cell contains notebook metadata, anything reading the notebook reads this)
    - EDIT markdown in 2nd cell manually
    - 3rd cell: This cell collection book metadata from Thoth book metadata service.
    - !! save notebook because quarto only renders what is (executed and) saved

**Essay**

- **collection003.qmd** (qmd means Quarto-Markdown)
    - Change text according to your collection

**Collections**

- **paintings_1xx.ipynb** ← use this name for your new collection, plus your initials instead of **xx** (CREATE via copying "**paintings.ipynb**")
    - *Simple:* Take the painting notebook, change collection name, number of items, dates, and RUN and SAVE. (see earlier instructions)
    - *Advanced:* Make own custom collection (own WikiData query), needs custom python. This will be added by support.
        - Create empty file named with your initials at end: **collection_2xx.ipynb**
        - You can get your code out of Wikidata query service directly: Move own Wikidata Query here via: code → Python → Copy/Paste
        - Copy-paste metadata cell from "Paintings" (1st cell contains notebook metadata, anything reading the notebook reads this)
        - Below: create another markdown cell and Copy-paste Query shortlink and SAVE. This is so your query can be referenced by support.
        - Add query python as a python cell. Run and save.

**Style**

- **_quarto.yml**
    - See line *theme:*

**Readme**

- **README.md**
- And for Quarto: **index.qmd**

**Include new files in TOC**

- **_quarto.yml**
    - At line 12 chapters: decide what goes into our publication
        - Include: paintings_1xx.ipynb, collection_2xx.ipynb
        - Exclude things you do not want to have in your publication (video, ...)
        - Make sure files are in right order:
            - Index.qmd
            - colophon.ipynb
            - collection003.qmd
            - paintings_1xx.ipynb
            - collection_2xx.ipynb

**Repository address**

- At line 21 repo-url, and SAVE. E.g., repo-url: https://github.com/NFDI4Culture/catalogue-003

## Note: Render and push back changes.

- Run notebooks, save notebooks.
- Terminal: quarto render
- Push back to repo via GitHub desktop, or GitHub in VSCode
- Add summary when committing to main
- And Push

# Review and Flight Check

These items will be covered in the class.

- Flight check: README; LICENCE; and add CONTRIBUTE file. Check other parts.
- Make a GitHub Release.
- Complete Zenodo deposit and release: We will add a PDF and an interoperable form. In this case a repository Zip file downloaded from the Github release
- Publish!