Run 3 Starterkit: Analysis Productions

Dylan Jaide White (they/them)

With thanks to Aidan Wiederhold, Chris Burr, Giulia Tuci, Ryun O'Neil

Introduction

- Today we'll be covering how to create ntuples using the **Analysis Productions (AP)** system
  - With a focus on some of the new features added for run 3
- This lesson is essentially a combination of:
  - Starterkit lesson on Analysis Productions
  - Hands-on session on using checks from the 8/11/21 EMTF meeting
- Except now using run 3 FEST data!
- We will:
  - Learn what Analysis Productions are & how they work
  - Create a simple production
  - Add some checks
  - Test the production with GitLab CI

Introduction
Disclaimers

- A few notes before we begin:
  - Parts of AP (primarily the checks) are still in development, and some specifics could change. Please see the docs for the most up-to-date info
  - Getting this to work for FEST data has required some extra steps compared to what will be needed for real in run 3, and some things aren't working like normal. I'll point these out as we go
- Now let's start!

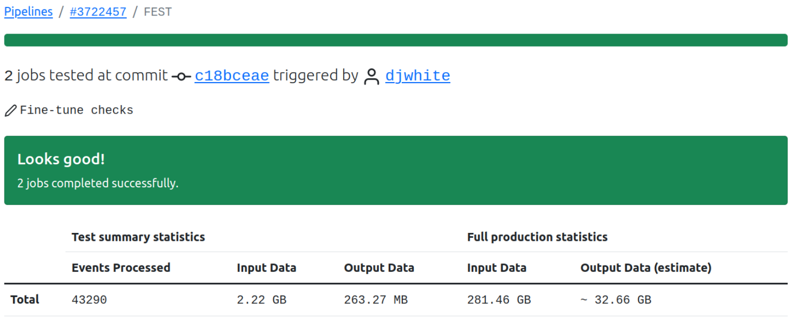
Overview of Analysis Productions

- Analysis Productions is a more streamlined/automated way to make ntuples
  - Automatically creates, submits, & manages grid jobs
- Compared to the old way of everyone creating their own ntuples & grid jobs, this has several advantages:
  - Automation - we don't have to monitor our own grid jobs! (or manually resubmit failed ones)
  - Validation - CI tests and liasons will help spot any mistakes before running
  - Preservation - all AP ntuples are automatically added to bookkeeping, stored centrally, and can always be reproduced
  - Sharing - all LHCb users can access ntuples from all productions, so different analyses using the same dataset can share

Overview of Analysis Productions

Example (1/3)

- Let's start by seeing what we'll be making today: link to AP web app



Pipelines / #3722457 / FEST

2 jobs tested at commit c18bceae triggered by djwhite

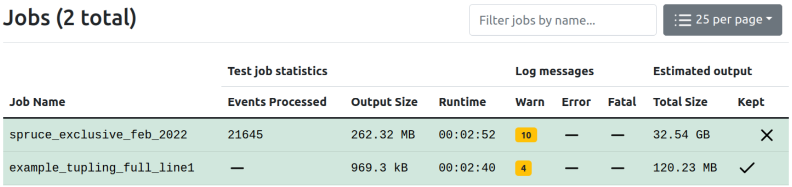
Fine-tune checks

Looks good!
2 jobs completed successfully.

	Test summary statistics			Full production statistics	
	Events Processed	**Input Data**	**Output Data**	**Input Data**	**Output Data (estimate)**
Total	43290	2.22 GB	263.27 MB	281.46 GB	~ 32.66 GB

- This is not actually a full production, but just the CI tests
  - This uses a small subset of data for fast testing before full submission
  - Later, the actual production would then run over the full dataset
- AP web app is an easy way to view info about both CI tests and full productions

Overview of Analysis Productions
Example (2/3)



Jobs (2 total)

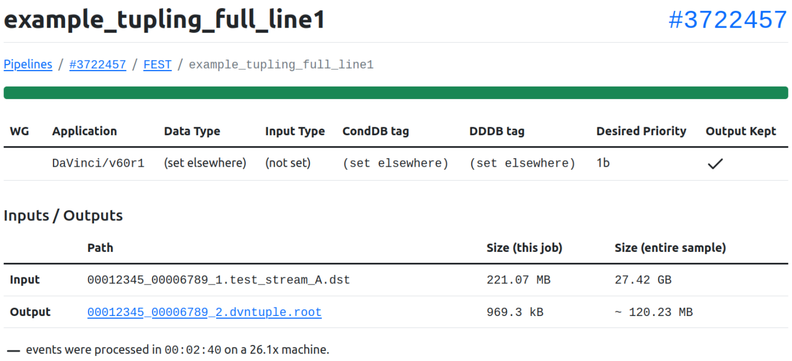
	Test job statistics			Log messages			Estimated output	
Job Name	Events Processed	Output Size	Runtime	Warn	Error	Fatal	Total Size	Kept
spruce_exclusive_feb_2022	21645	262.32 MB	00:02:52	10	—	—	32.54 GB	✗
example_tupling_full_line1	—	969.3 kB	00:02:40	4	—	—	120.23 MB	✓

- Here we see two jobs were created:
  - One for sprucing, one for ntupling (DaVinci)
- Usually, you won't need to include sprucing in your production
  - You would probably use pre-made DSTs as inputs for DaVinci
- This is just needed for this FEST data to work today!
- For a normal production, this is usually for multiple DaVinci jobs
  - For different years, polarities, etc.

Overview of Analysis Productions

Example (3/3)

- Click on the "example_tupling..." job to see more about it



example_tupling_full_line1 #3722457

Pipelines / #3722457 / FEST / example_tupling_full_line1

WG	Application	Data Type	Input Type	CondDB tag	DDDB tag	Desired Priority	Output Kept
	DaVinci/v60r1	(set elsewhere)	(not set)	(set elsewhere)	(set elsewhere)	1b	✓

Inputs / Outputs

	Path	Size (this job)	Size (entire sample)
Input	00012345_00006789_1.test_stream_A.dst	221.07 MB	27.42 GB
Output	00012345_00006789_2.dvntuple.root	969.3 kB	~ 120.23 MB

— events were processed in 00:02:40 on a 26.1x machine.

- Here you can see lots of info about this job:
  - Input & output info (+ download for output file)
  - Results of checks (more about these later...)
  - In-browser TTree viewer - very handy!

Hands on (part 1)

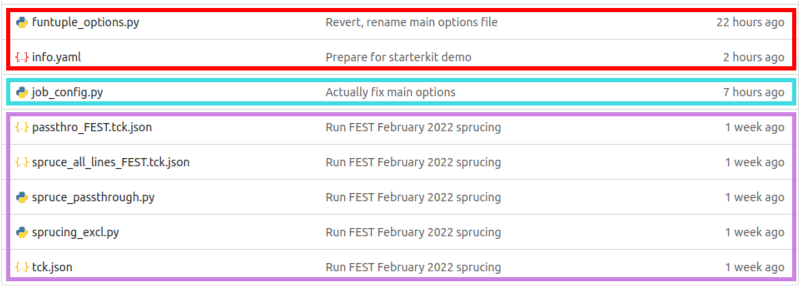
Cloning AP repo

- Let's jump right into making this
- First we need to clone the AnalysisProductions repo
  - `git clone ssh://git@gitlab.cern.ch:7999/lhcb-datapkg/AnalysisProductions.git`
  - `git clone https://gitlab.cern.ch/lhcb-datapkg/AnalysisProductions.git`
  - etc.
- `cd AnalysisProductions`
- Now checkout the branch we'll be starting from:
  - `git checkout -b ${USER}/starterkit-run3-fest --no-track origin/djwhite/starterkit-run3-fest`
  - Changing the name and not tracking is so everyone can push their branch independently later on
- We'll be working with the production called `FEST`

Hands on (part 1)

Understanding the files

- Let's see what files are in this production:



funtuple_options.py	Revert, rename main options file	22 hours ago
info.yaml	Prepare for starterkit demo	2 hours ago
job_config.py	Actually fix main options	7 hours ago
passthro_FEST.tck.json	Run FEST February 2022 sprucing	1 week ago
spruce_all_lines_FEST.tck.json	Run FEST February 2022 sprucing	1 week ago
spruce_passthrough.py	Run FEST February 2022 sprucing	1 week ago
sprucing_excl.py	Run FEST February 2022 sprucing	1 week ago
tck.json	Run FEST February 2022 sprucing	1 week ago

- Split up into 3 groups:
  - Top (red): main files needed for an AP. **Main focus of this lesson!**
    - DaVinci options, and a YAML configuration file for AP
  - Middle (cyan): a second options file for DaVinci, with extra config needed to make FEST data work
  - Bottom (purple): everything for the sprucing job

Hands on (part 1)
FunTuple options

- First let's look at `funtuple_options.py` (view on GitLab)
- Hopefully this looks very familiar to you - if you followed the FunTuple lesson on Wednesday from Abhijit, Davide, & Martina, you essentially already wrote this yourself!
- Different decay ($K_s^0 \to \pi^+\pi^-$), but very similar otherwise
- Other than the decay, the only real differences are some small syntax changes
    - *e.g.* `branches` instead of `fields` in the `FunTuple_Particles` constructor
- Reason: AP can only used released versions of LHCb software
    - The most recent release of DaVinci is v60r1, which was released before this syntax was changed
    - Yet another reason to start writing any functors you want sooner rather than later!

Hands on (part 1)
YAML configuration file

- Now let's look at `info.yaml` (view on GitLab)
- This is the file that defines what jobs the production will create, and what files they each need
- We can see the sprucing job is in here already
- There is also the `defaults` job name
  - Keyword - anything under defaults will be applied to all other jobs
- Note the `automatically_configure` option
  - You would normally use this to automatically set things like DDDB/CondDB tags, data types, etc.
  - But this isn't working for FEST data right now
  - So for now, this is turned off, and those values are already set manually in the second options file I mentioned earlier

Hands on (part 1)

`lb-ap` commands

- For working with productions locally in this repo, we can use `lb-ap` commands
  - Remember to get a proxy if you haven't already: `lhcb-proxy-init`
- Try: `lb-ap list FEST`
  - This will list all the jobs in this production
  - Currently, this is only the pre-existing sprucing job
- So let's add a DaVinci job!

Hands on (part 1)
Adding a DaVinci job (1/2)

- This takes a very similar form to the sprucing job that's already there
- We need to specify the following:
    - `application` - the relevant DaVinci release (here v60r1)
    - `output` - name for a `.root` ntuple
    - `input` - usually, you would specify a `.dst` from bookkeeping with `bk_query`, but for the FEST data we will instead use the output from the sprucing job directly
    - `options` - the two DaVinci options files we saw earlier

Hands on (part 1)
Adding a DaVinci job (2/2)

- Once we're done we should have something like this:

```
example_tupling_full_line1:
    application: DaVinci/v60r1
    output: dvntuple.root
    input:
        job_name: spruce_exclusive_feb_2022
        filetype: TEST_STREAM_A.DST
    options:
        - funtuple_options.py
        - job_config.py
```

Hands on (part 1)

Local testing with `lb-ap test`

- This is usually when I would say: "Now we can use `lb-ap test` to run this job locally, to make sure it works before pushing your branch"
- Unfortunately, there's been an issue running local tests with the FEST data that we haven't been able to fix yet
- The best we can do for now is validating the options:
  - `lb-ap validate FEST`
- If that comes back all OK, let's move straight to pushing the branch:
  - `git add FEST/info.yaml`
  - `git commit -m "<commit message here>"`
  - `git push origin ${USER}/starterkit-run3-fest`
- Now seems like a good opportunity to take a short break!

Hands on (part 1)

Looking at our pipelines on the web app

- After pushing your branch, go to the pipelines section of the web app
- Your branch should be there - click on it to see how it's going!
- Hopefully everything is green, and ran correctly!

- It should look very similar to what we saw earlier when we looked at the final results
- Except, it will be missing any checks - we didn't add any!
- So let's look at how to do that now

Checks

Overview

- The **checks** system is a new addition to AP
- CI tests are very good at spotting code that breaks/crashes
- But not so good at identifying subtle issues with data quality, or missing information that you might want for your analysis

- Idea: to be able to easily define requirements that the data in the finished ntuple should satisfy
- These are defined in the `info.yaml` as well

- Currently, six types of checks, including:
  - Minimum number of entries (per unit luminosity)
  - Require certain branches to exist
  - Create assorted histograms & verify certain properties

Checks

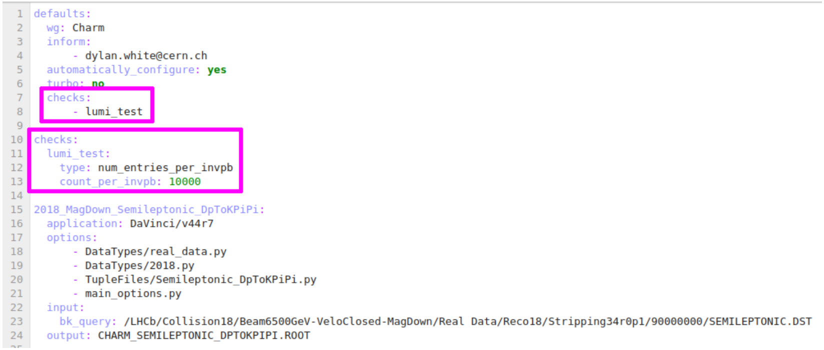
Getting started

- Checks are added to a production's `info.yaml` in 2 stages:
- First: define your check
  - These go under a `checks` keyword job name (similar to `default`)
  - You give your check a name, then define its type & parameters
  - Each check type has different parameters - see the docs for what each one needs
- Second: add your check to the relevant jobs
  - List check names under a `checks` option in each job
  - For today, we only want to add checks to the job running DaVinci

Checks

Example

- Example from a different production (using real run 2 data): a simple check that's been added to an `info.yaml` file:



```yaml
defaults:
  wg: Charm
  inform:
      - dylan.white@cern.ch
  automatically_configure: yes
  turbo: no
  checks:
      - lumi_test

checks:
  lumi_test:
    type: num_entries_per_invpb
    count_per_invpb: 10000

2018_MagDown_Semileptonic_DpToKPiPi:
  application: DaVinci/v44r7
  options:
      - DataTypes/real_data.py
      - DataTypes/2018.py
      - TupleFiles/Semileptonic_DpToKPiPi.py
      - main_options.py
  input:
    bk_query: /LHCb/Collision18/Beam6500GeV-VeloClosed-MagDown/Real Data/Reco18/Stripping34r0p1/90000000/SEMILEPTONIC.DST
  output: CHARM_SEMILEPTONIC_DPTOKPIPI.ROOT
```

- As you can see, it's only 6 extra lines to add this check!
- Now let's write one ourselves which will work for the FEST data

Hands on (part 2)
Adding your first check (1/2)

- Let's start with the simplest one: `num_entries`
  - I find it's helpful to have the docs open for this

- First: let's add a new section (base level) called `checks`
- Within that, we give our new check a name - anything you want
  - Something descriptive is ideal - I'll use `require_100_entries`
- Indented another level, we can now start to define our new check
  - Every check must have a `type` provided - for this one, `num_entries`
  - You can then check the docs to see what else is needed for that type
  - Here, we only need `count`, which we'll set to 100
- It should look something like this:

```yaml
checks:
    at_least_100_entries:
        type: num_entries
        count: 100
```

Hands on (part 2)

Adding your first check (2/2)

- All that remains is to tell it which jobs to use this check on
  - If you define a check but never use it, the YAML file parsing won't be successful - all checks defined must be used by at least one job
- We only want to add checks to the tupling job
- Under `example_tupling...` (or whatever you called it), add a `checks` option
- Within that, create a list (using `-`) and add your check's name

```yaml
example_tupling_full_line1:
    application: DaVinci/v60r1
    ...
    checks:
        - at_least_100_entries
```

- You can check your YAML again with `lb-ap validate FEST`

Aside: `lb-ap check`

- Just like earlier with `lb-test`, we can't currently test this locally
- If we could: now is when I would show you `lb-ap check`
- This only runs the checks, but requires a `.root` file from a previous `test` command
- This means you don't have to wait for DaVinci every time if you've only changed your checks!

- Once run 3 arrives, this will work!
- But for today, we'll have to skip this & move straight on to adding more checks

Hands on (part 2)

Creating a branches_exist check

- Another simple check type is branches_exist
  - Only one required parameter: branches
- Check fails if any of these branches are not present in the final ntuple

```yaml
checks:
    ...
    Ks0_branches:
        type: branches_exist
        branches:
            - KS0_M
            - KS0_PX
            - KS0_PY
            - KS0_PZ
```

- Then add the check name to the checks list in the ntupling job
- Advanced: can try using Jinja templating to create the list of branches using loops instead of writing out each entry (see docs)

Hands on (part 2)

Creating a `range` check (1/2)

- When we looked on the web app earlier, we saw some histograms under the checks section
- Let's create a 1D histogram using a `range` check
- Looking again at the docs, we can see that only two things are required for a range check:
  - `expression`: what the histogram will be of
    - Let's use `KS0_M`
  - `limits`: upper/lower bounds for the x-axis
    - `min: 460`
    - `max: 540`
- The other options (marked with *) are optional
- But let's try out `exp_mean` and `mean_tolerance`
  - `exp_mean: 497`
  - `mean_tolerance: 5.0`
  - This check will fail if the histogram's mean is not in the range 497 ± 5
    - Failed checks cause the full CI test to fail, so mistakes can be caught!

Hands on (part 2)

Creating a range check (2/2)

- Want to end up with something like this:

```yaml
checks:
    ...
    Ks0_mass_hist:
        type: range
        expression: KS0_M
        limits:
            min: 460
            max: 540
        exp_mean: 497
        mean_tolerance: 5
```

- Again, remember to add the name of this new check to the `checks` list in the ntupling job!

Hands on (part 2)

Creating a second `range` check (1/2)

- Let's try out some of the other features of `range` checks using a different histogram
- The `expression` parameter can be a combination of variables
  - *e.g.* `(KS0_PX**2 + KS0_PY**2 + KS0_PZ**2)**0.5`
- If using this: reasonable upper/lower bounds are 0/120,000
- Custom number of bins: set `n_bins` to an integer on [2,100]
- Blinding:
  - In responses to last year's surveys in WGs, this was one of the most requested features
  - Can add multiple ranges to blind
  - List of max/min limits - pick a range to blind
  - Advanced: try adding multiple blinding ranges

Hands on (part 2)

Creating a second range check (2/2)

- Here are the values I'll use:

```
checks:
    ...
    Ks0_mom_hist:
        type: range
        expression: (KS0_PX**2 + KS0_PY**2 + KS0_PZ**2)**0.5
        limits:
            min: 0
            max: 120000
        n_bins: 40
        blind_ranges:
            -
                min: 40000
                max: 50000
```

Hands on (part 2)

Creating a 2D hist: range_nd check (1/2)

- The range_nd check can be used to make 2D (or 3D) hists
- Syntax similar to range, but with space for multiple axes

```
checks:
    ...
    Ks0_mom_xy_hist:
        type: range_nd
        expressions:
            x: KS0_PX
            y: KS0_PY
        limits:
            x:
                min: -3000
                max: 3000
...
```

Hands on (part 2)

Creating a 2D hist: `range_nd` check (2/2)

```
...
        y:
            min: -3000
            max: 3000
    n_bins:
        x: 20
        y: 20
```

- Advanced: try adding one or more blinding ranges
- Once you think you've finished, try validating your production again
  - `lb-ap validate FEST`
- If that works, commit your changes & push your branch again
- While we wait for that to run...

Aside: Other check types

- Two more check types we haven't used here today:
  - `num_entries_per_invpb`: requires at least a certain number of events per pb^{-1} of lumi
    - Only works with real data
  - `range_bkg_subtracted`: like the 1D `range` check, but can perform simple background subtraction (without any fits)
- All the details on how to use these are in the docs

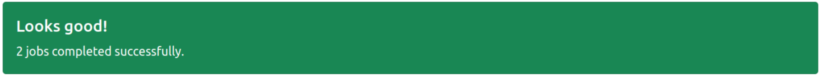
Aside: Offline HLT2 monitoring

- We are currently working on plans to use AP for an automated offline HLT2 monitoring system
- With the checks feature, we are able to perform checks on HLT2 output, including creating histograms
- Aim: to set up a system that lets us automatically run regular APs on early run 3 data, and display the plots somewhere convenient
    - Ideally this will be as easy-to-use as possible, so that everyone is able to quickly set up monitoring for their own lines/channels

Hands on (part 2)

Viewing check results in CI

- Now let's look at the checks in the web app!
- If your pipelines still haven't finished, use this one for now: link
- Hopefully all the checks have passed!

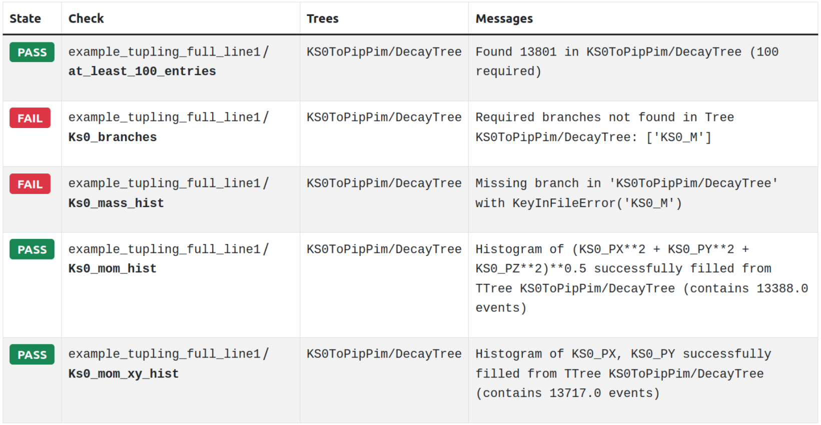


Looks good!

2 jobs completed successfully.

What happens when checks fail?

- Here's a different pipeline where the checks didn't pass: link



State	Check	Trees	Messages
PASS	example_tupling_full_line1/ **at_least_100_entries**	KS0ToPipPim/DecayTree	Found 13801 in KS0ToPipPim/DecayTree (100 required)
FAIL	example_tupling_full_line1/ **Ks0_branches**	KS0ToPipPim/DecayTree	Required branches not found in Tree KS0ToPipPim/DecayTree: ['KS0_M']
FAIL	example_tupling_full_line1/ **Ks0_mass_hist**	KS0ToPipPim/DecayTree	Missing branch in 'KS0ToPipPim/DecayTree' with KeyInFileError('KS0_M')
PASS	example_tupling_full_line1/ **Ks0_mom_hist**	KS0ToPipPim/DecayTree	Histogram of (KS0_PX**2 + KS0_PY**2 + KS0_PZ**2)**0.5 successfully filled from TTree KS0ToPipPim/DecayTree (contains 13388.0 events)
PASS	example_tupling_full_line1/ **Ks0_mom_xy_hist**	KS0ToPipPim/DecayTree	Histogram of KS0_PX, KS0_PY successfully filled from TTree KS0ToPipPim/DecayTree (contains 13717.0 events)

- The messages should help to identify what's wrong
- In this case: K_s^0 mass branch was missing from ntuple

Final remarks

- Today we have covered how to:
  - Create a simple analysis production
  - Add checks to a production
  - View the results of checks using the AP web app
- But there's still a lot of work ongoing
  - And a few bugs still to squash
- We hope you will use checks in your productions for run 3!
  - They should hopefully double as both an easy way to check your data looks as you expect, and a way to automatically monitor your lines
- Your feedback is very useful to us!
  - The system need to be tested to spot possible bugs and improve the user experience
  - Are there any other types of checks that you would find useful?
- Thanks for coming! I hope you've enjoyed the Starterkit this week!
- And thanks to all the organisers!