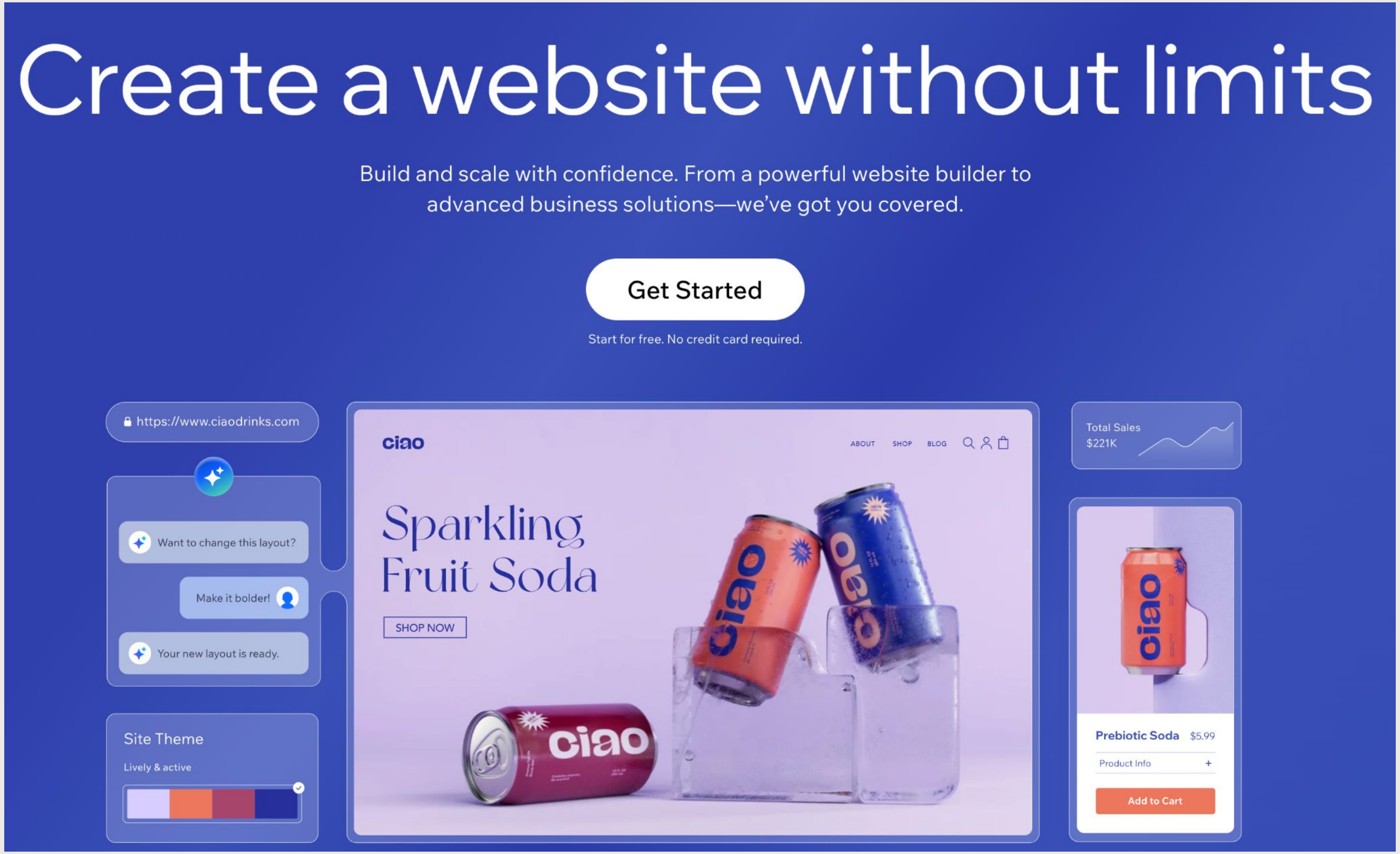
Refactoring DAGs
From Duplication to Delightful Efficiency with a Centralized Library

Gil Reich
Data Engineer for Data Science at Wix

Wix


Create a website without limits
Build and scale with confidence. From a powerful website builder to advanced business solutions—we've got you covered.
Get Started
Start for free. No credit card required.
https://www.ciaodrinks.com
Want to change this layout?
Make it bolder!
Your new layout is ready.
Site Theme
Lively & active
ciao
ABOUT
SHOP
BLOG
Sparkling Fruit Soda
SHOP NOW
Total Sales
$221K
Prebiotic Soda
$5.99
Product Info
Add to Cart

The **Wix website builder** offers a complete solution from **enterprise-grade infrastructure** and **business features** to **advanced SEO** and **marketing tools**–enabling anyone to create and grow online.

Wix

10%

of websites are created on Wix

220M

Sites were built via Wix

5000

People work at Wix

Wix & Airflow

1500

Airflow DAGs

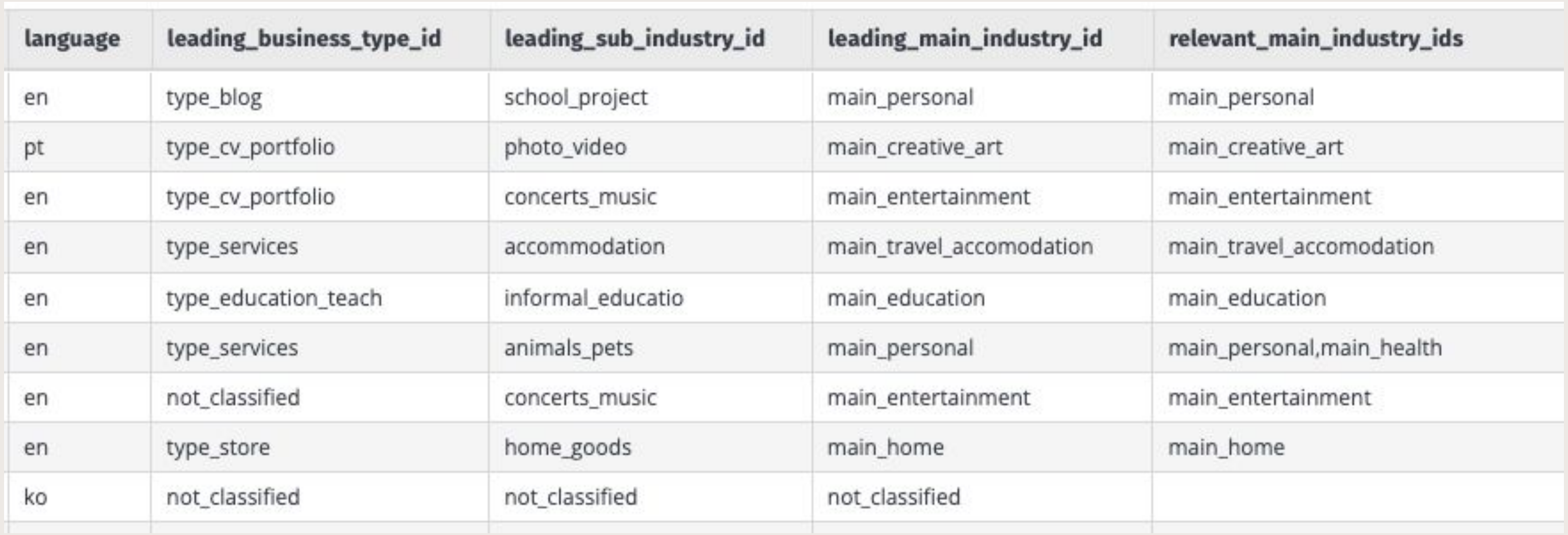
12K

Airflow tasks

5500

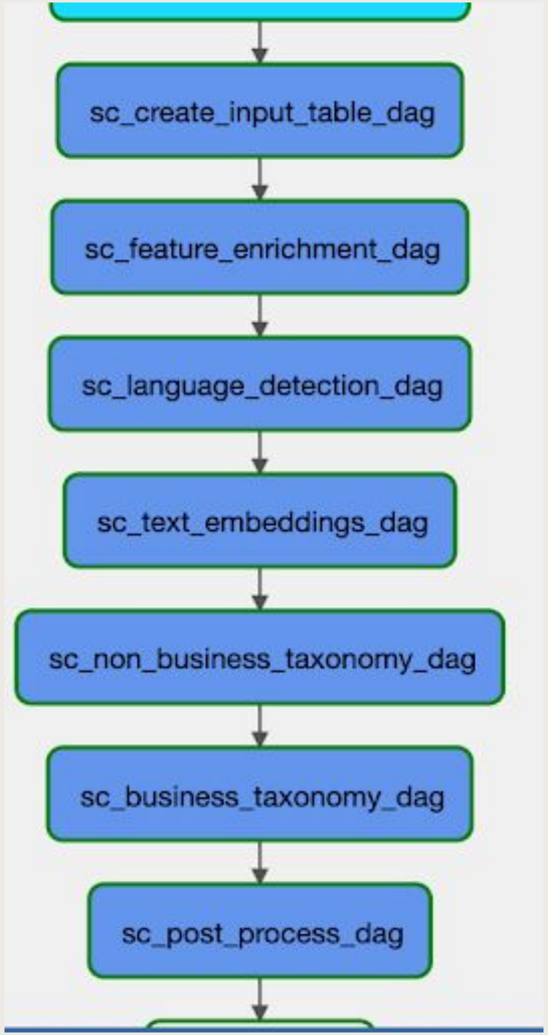
Daily DAG runs

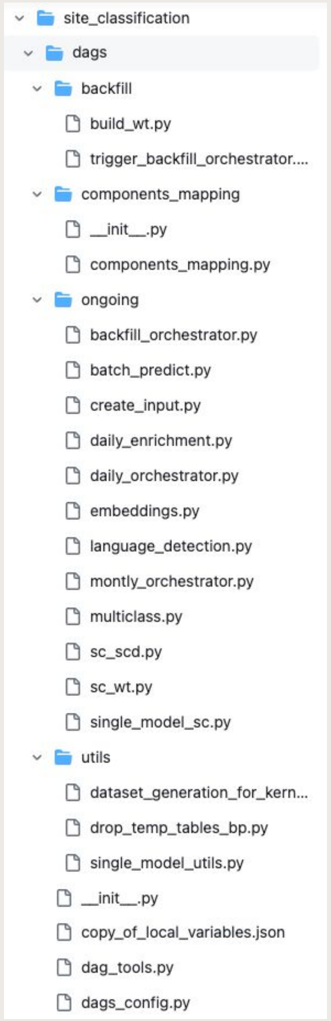
Site Classification



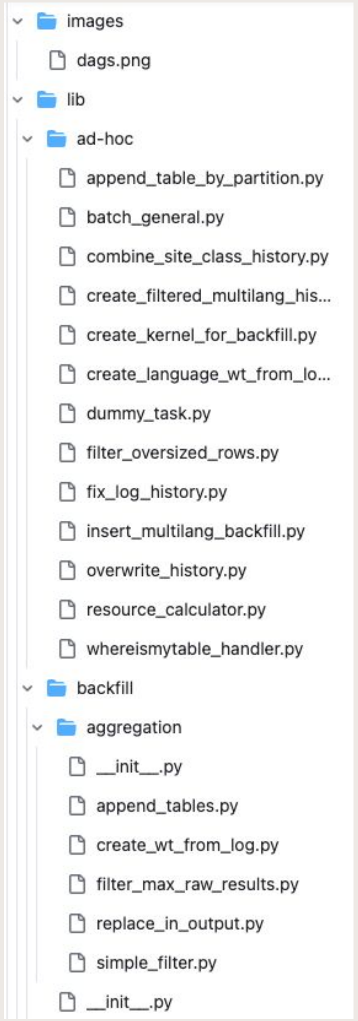
language	leading_business_type_id	leading_sub_industry_id	leading_main_industry_id	relevant_main_industry_ids
en	type_blog	school_project	main_personal	main_personal
pt	type_cv_portfolio	photo_video	main_creative_art	main_creative_art
en	type_cv_portfolio	concerts_music	main_entertainment	main_entertainment
en	type_services	accommodation	main_travel_accomodation	main_travel_accomodation
en	type_education_teach	informal_educatio	main_education	main_education
en	type_services	animals_pets	main_personal	main_personal,main_health
en	not_classified	concerts_music	main_entertainment	main_entertainment
en	type_store	home_goods	main_home	main_home
ko	not_classified	not_classified	not_classified	

The Orchestrators & the 7 DAGs

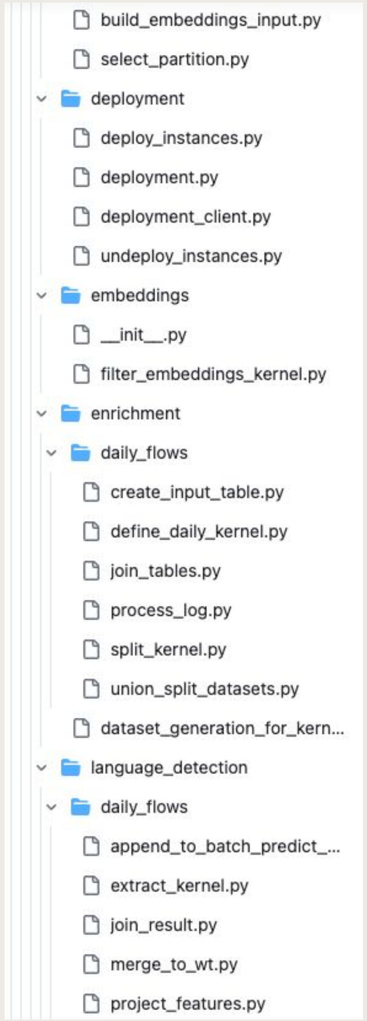

sc_create_input_table_dag
sc_feature_enrichment_dag
sc_language_detection_dag
sc_text_embeddings_dag
sc_non_business_taxonomy_dag
sc_business_taxonomy_dag
sc_post_process_dag



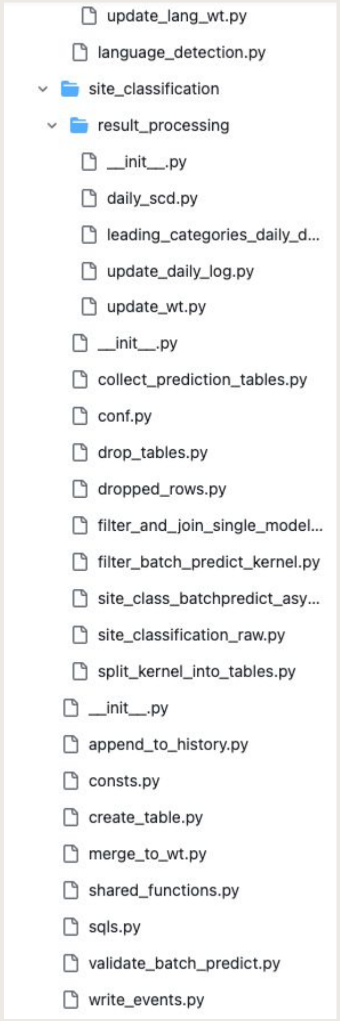
```
site_classification
  dags
    backfill
      build_wt.py
      trigger_backfill_orchestrator....
    components_mapping
      __init__.py
      components_mapping.py
    ongoing
      backfill_orchestrator.py
      batch_predict.py
      create_input.py
      daily_enrichment.py
      daily_orchestrator.py
      embeddings.py
      language_detection.py
      montly_orchestrator.py
      multiclass.py
      sc_scd.py
      sc_wt.py
      single_model_sc.py
    utils
      dataset_generation_for_kern...
      drop_temp_tables_bp.py
      single_model_utils.py
    __init__.py
    copy_of_local_variables.json
    dag_tools.py
    dags_config.py
```



```
images
  dags.png
lib
  ad-hoc
    append_table_by_partition.py
    batch_general.py
    combine_site_class_history.py
    create_filtered_multilang_his...
    create_kernel_for_backfill.py
    create_language_wt_from_lo...
    dummy_task.py
    filter_oversized_rows.py
    fix_log_history.py
    insert_multilang_backfill.py
    overwrite_history.py
    resource_calculator.py
    whereismytable_handler.py
  backfill
    aggregation
      __init__.py
      append_tables.py
      create_wt_from_log.py
      filter_max_raw_results.py
      replace_in_output.py
      simple_filter.py
    __init__.py
```



```
    build_embeddings_input.py
    select_partition.py
  deployment
    deploy_instances.py
    deployment.py
    deployment_client.py
    undeploy_instances.py
  embeddings
    __init__.py
    filter_embeddings_kernel.py
  enrichment
    daily_flows
      create_input_table.py
      define_daily_kernel.py
      join_tables.py
      process_log.py
      split_kernel.py
      union_split_datasets.py
    dataset_generation_for_kern...
  language_detection
    daily_flows
      append_to_batch_predict_...
      extract_kernel.py
      join_result.py
      merge_to_wt.py
      project_features.py
```



```
      update_lang_wt.py
    language_detection.py
  site_classification
    result_processing
      __init__.py
      daily_scd.py
      leading_categories_daily_d...
      update_daily_log.py
      update_wt.py
    __init__.py
    collect_prediction_tables.py
    conf.py
    drop_tables.py
    dropped_rows.py
    filter_and_join_single_model...
    filter_batch_predict_kernel.py
    site_class_batchpredict_asy...
    site_classification_raw.py
    split_kernel_into_tables.py
  __init__.py
  append_to_history.py
  consts.py
  create_table.py
  merge_to_wt.py
  shared_functions.py
  sqls.py
  validate_batch_predict.py
  write_events.py
```

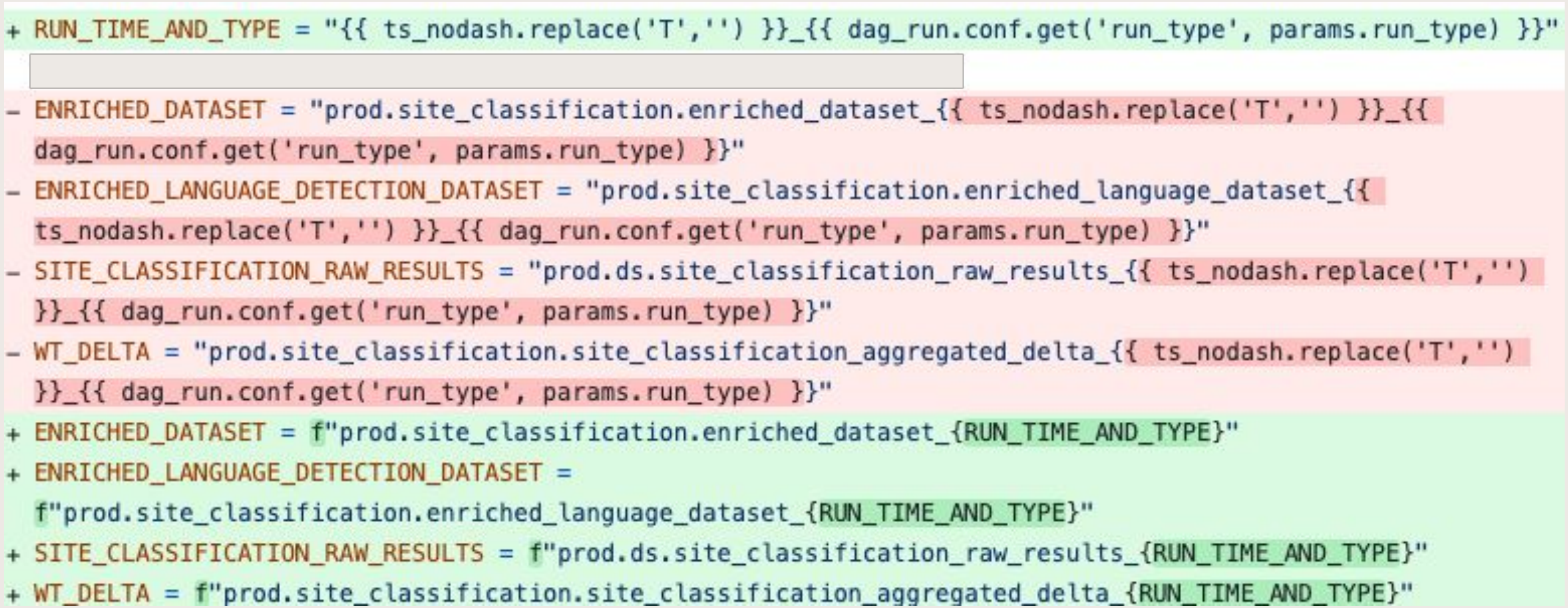
Our Guiding Principles:

1. Reduce duplicate code
2. Abstract away ugliness and complexity
3. Short and simple functions
4. Classes (not everyone agrees about this)
5. Separate business logic, config, technical implementation
6. Separate specific & shared
7. Start small

What we did:
The 6 problems

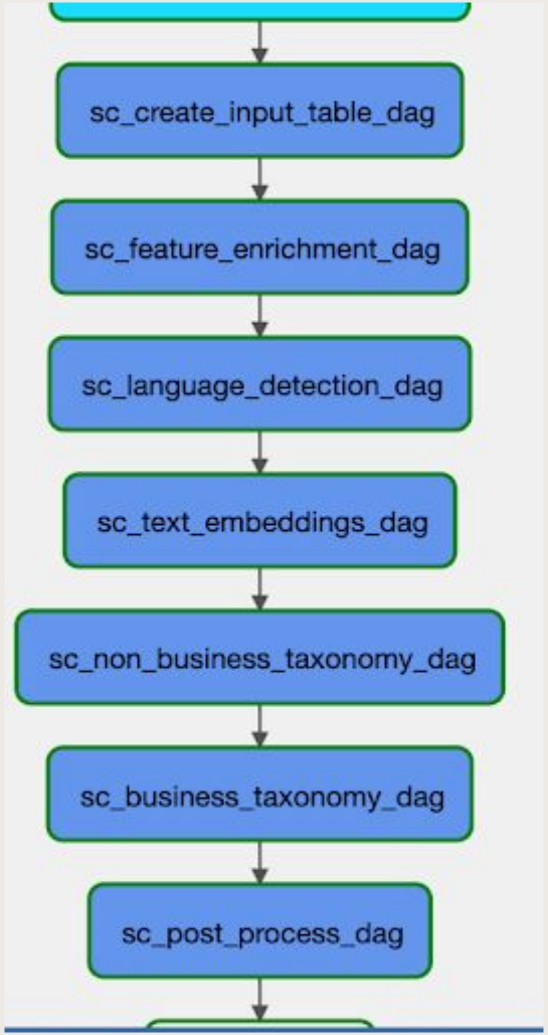
Problem 1:
Near-Duplicate Code

Started with Simple Replacements



```diff
+ RUN_TIME_AND_TYPE = "{{ ts_nodash.replace('T','') }}_{{ dag_run.conf.get('run_type', params.run_type) }}"

- ENRICHED_DATASET = "prod.site_classification.enriched_dataset_{{ ts_nodash.replace('T','') }}_{{
  dag_run.conf.get('run_type', params.run_type) }}"
- ENRICHED_LANGUAGE_DETECTION_DATASET = "prod.site_classification.enriched_language_dataset_{{
  ts_nodash.replace('T','') }}_{{ dag_run.conf.get('run_type', params.run_type) }}"
- SITE_CLASSIFICATION_RAW_RESULTS = "prod.ds.site_classification_raw_results_{{ ts_nodash.replace('T','')
  }}_{{ dag_run.conf.get('run_type', params.run_type) }}"
- WT_DELTA = "prod.site_classification.site_classification_aggregated_delta_{{ ts_nodash.replace('T','')
  }}_{{ dag_run.conf.get('run_type', params.run_type) }}"
+ ENRICHED_DATASET = f"prod.site_classification.enriched_dataset_{RUN_TIME_AND_TYPE}"
+ ENRICHED_LANGUAGE_DETECTION_DATASET =
  f"prod.site_classification.enriched_language_dataset_{RUN_TIME_AND_TYPE}"
+ SITE_CLASSIFICATION_RAW_RESULTS = f"prod.ds.site_classification_raw_results_{RUN_TIME_AND_TYPE}"
+ WT_DELTA = f"prod.site_classification.site_classification_aggregated_delta_{RUN_TIME_AND_TYPE}"
```

The Orchestrators & the 7 DAGs


sc_create_input_table_dag
sc_feature_enrichment_dag
sc_language_detection_dag
sc_text_embeddings_dag
sc_non_business_taxonomy_dag
sc_business_taxonomy_dag
sc_post_process_dag

Combined Code (Orchestrators)

Before:

1. daily_orchestrator.py
2. monthly_orchestrator.py
3. backfill_orchestrator.py

Orchestrators section of sc_config.py

```
DagConfKeys.ORCHESTRATORS: {
   "run_types": {
       "daily": {DagConfKeys.SCHEDULE: "30 3 * * *"},
       "monthly": {DagConfKeys.SCHEDULE: "0 11 1 * *"},
       "backfill": {DagConfKeys.SCHEDULE: None}
   }
}
```

Combined Code (Orchestrators)

Before:

1. daily_orchestrator.py
2. monthly_orchestrator.py
3. backfill_orchestrator.py

After:

1. orchestrator.py
2. sc_config.py

Combined Code (Internal DAGs)

Before:

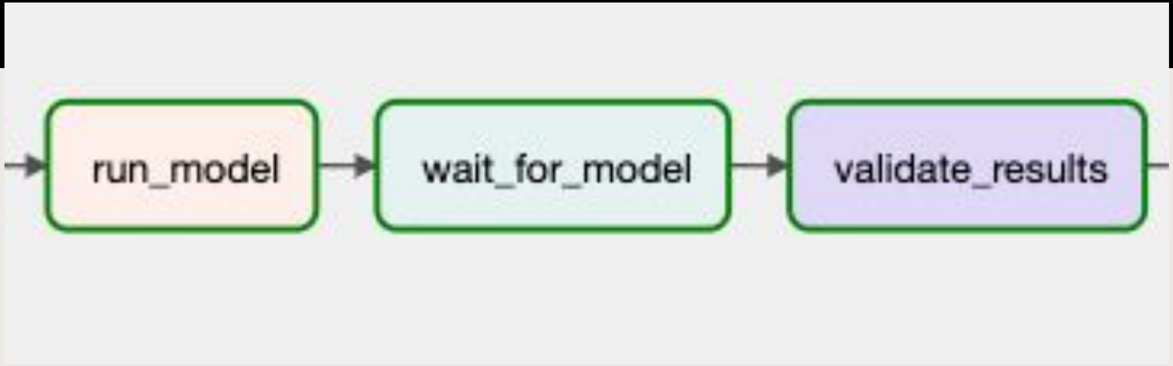
1. create_input.py
2. feature_enrichment.py
3. language_detection.py
4. embeddings.py
5. non_business_taxonomy.py
6. business_taxonomy.py
7. post_process.py

After:

1. ds_dag.py
2. sc_create_dags.py
3. sc_config.py

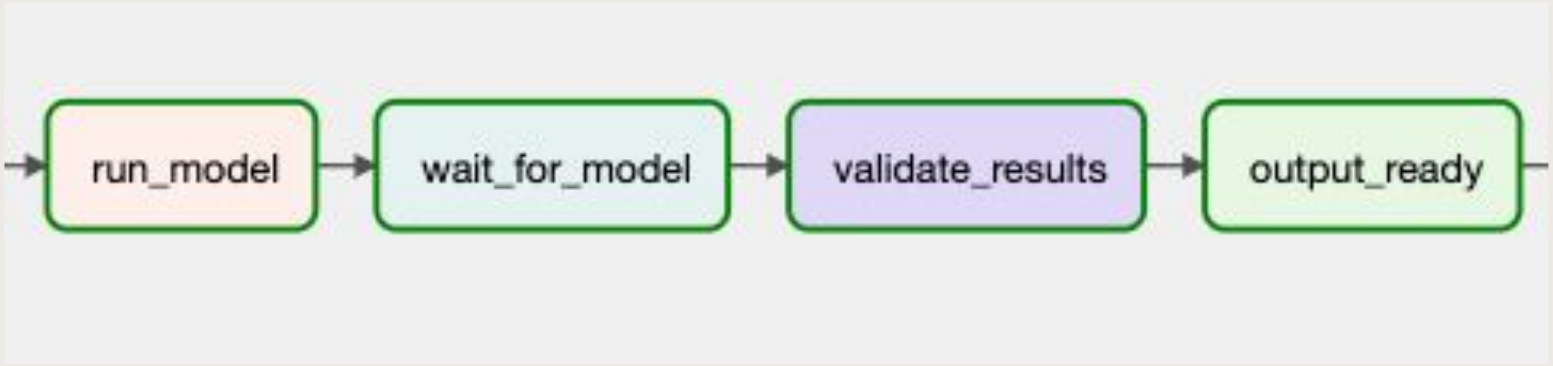
Group Tasks: Before

```
run_model = PythonOperator(parameters)
wait_for_model = PythonSensor(parameters)
validate_model = PythonOperator(parameters)


run_model >> wait_for_model >> validate_model
```



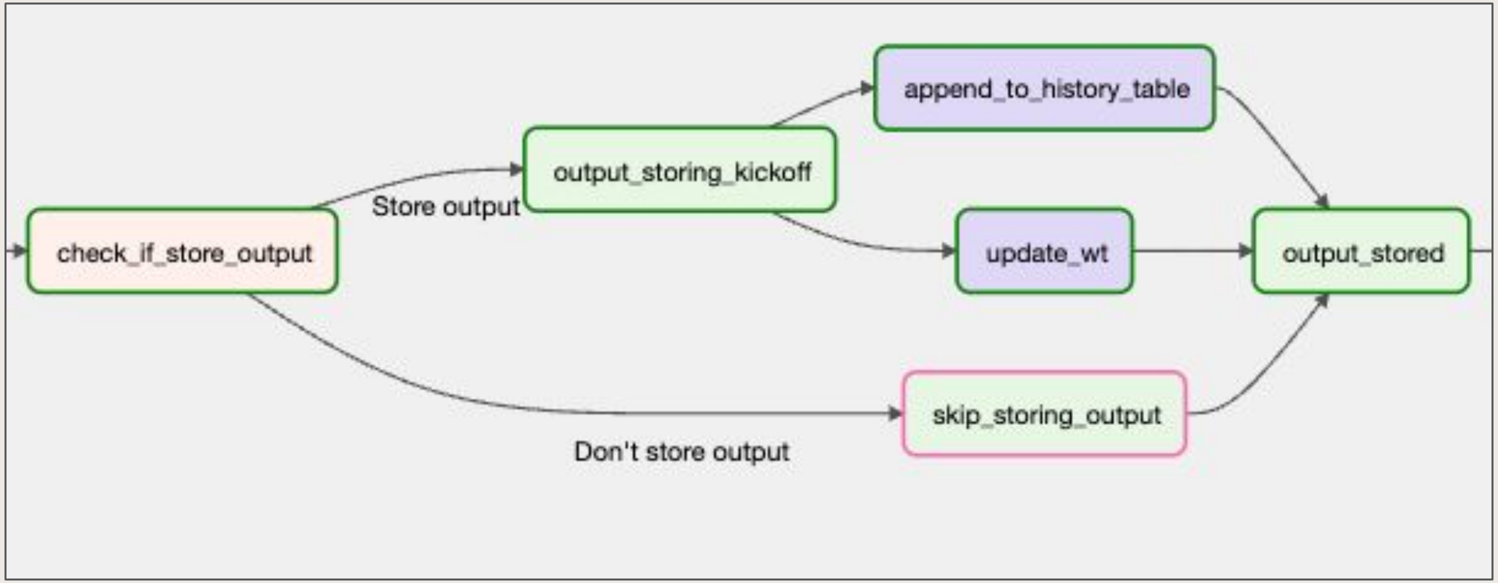
Group Tasks: After

```
ds_dag.new_tasks_run_model()
```



New Tasks: Store Results

```
ds_dag.new_tasks_store_results()
```



DS DAG Class Library

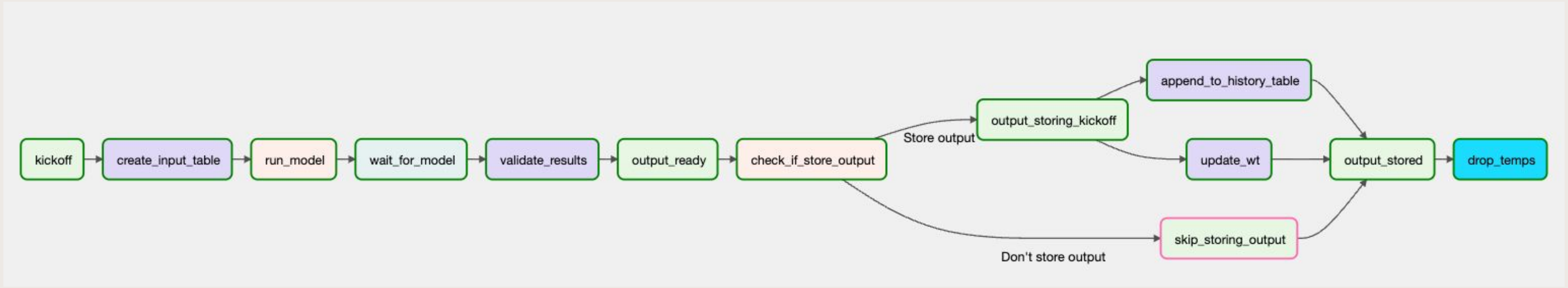
```
class DSDagManager (1 instance per project)

class DSDag (1 instance per internal DAG)

class Orchestrator(DSDag) (1 instance per Orchestrator)
```

sc_config.py

```
DagIDs.NON_BUSINESS_TAXONOMY: {
    DagConfKeys.ESSENCE: "Classifies the site as one of the
following: normal, low content, template or coming soon.",
    DagConfKeys.INPUT_DAG: DagIDs.FEATURE_ENRICHMENT,
    DagConfKeys.MODEL: "ds-sc-out-of-taxonomy",
    DagConfKeys.SHORT_NAME: "non_business_taxonomy",
    DagConfKeys.WT: "prod.ds.sc_non_business_taxonomy_wt",
    DagConfKeys.WT_DATE_COLUMN: 'revision_date,execution_date',
    DagConfKeys.HISTORY_FIELDS:
f"{COMMON_HISTORY_FIELDS},coming_soon_proba,low_content_proba,nor
mal_proba,predict,template_proba",
},
```

Putting it all together: sc_create_dags.py

```
ds_dag = ds_dag_manager.create_dag(DagIDs.NON_BUSINESS_TAXONOMY)
ds_dag.new_task_create_table(sql_id=XXX)
ds_dag.new_tasks_run_model()
ds_dag.new_tasks_store_results()
```



Average Internal DAG

Before:

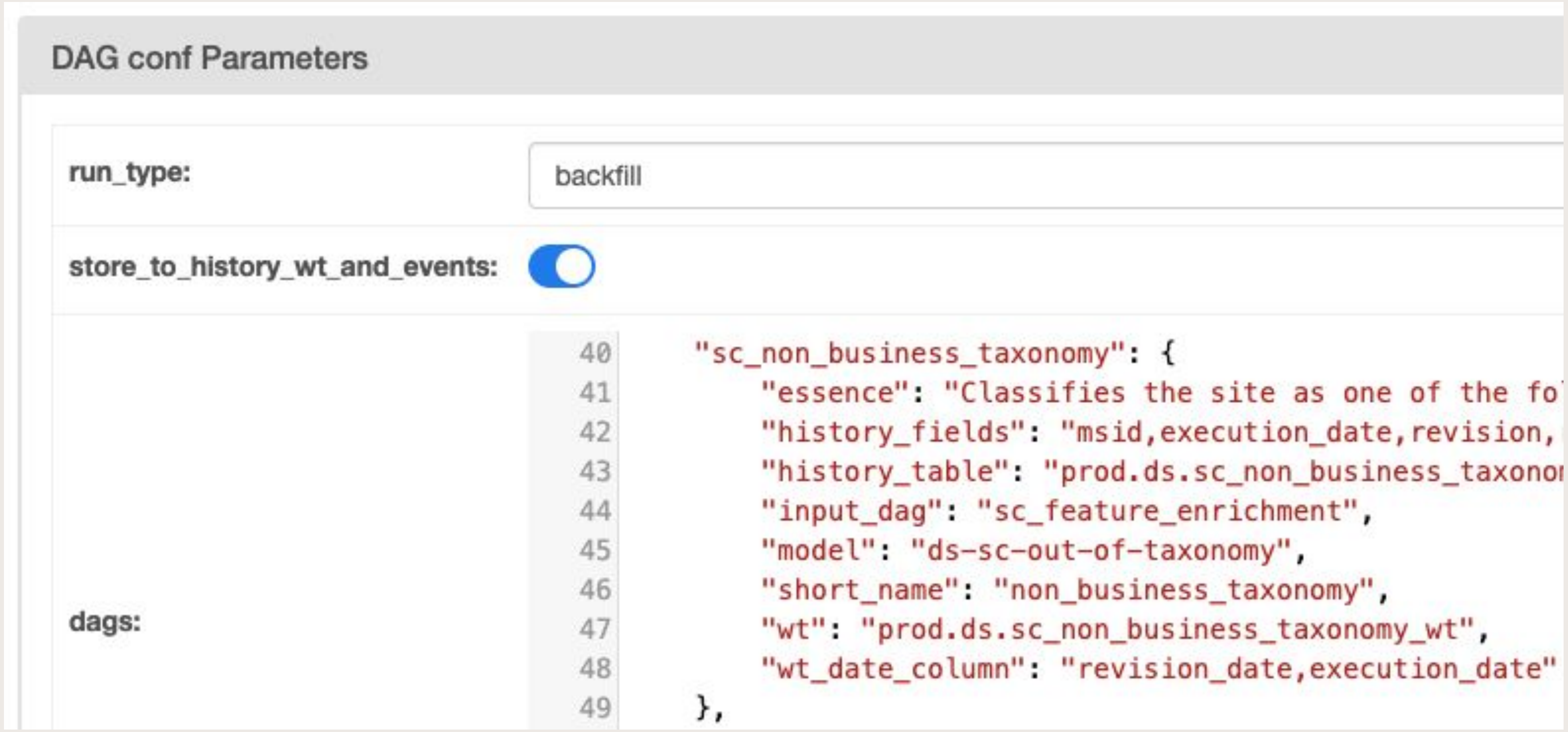
1. DAG file (55 lines)
2. config file (8 lines)
3. Airflow variable (5 lines)
4. extract_kernel (65 lines)
5. append_to_history (57 lines)
6. merge_to_wt (64 lines)

After:

1. sc_create_dags (8 lines)
2. sc_config.py (7 lines)
3. sc_sqls.py (10 lines)
4. (plus shared code in DS DAG Framework)

Problem 2:
Ad-Hoc Runs

Ad-hoc runs



DAG conf Parameters

run_type: backfill

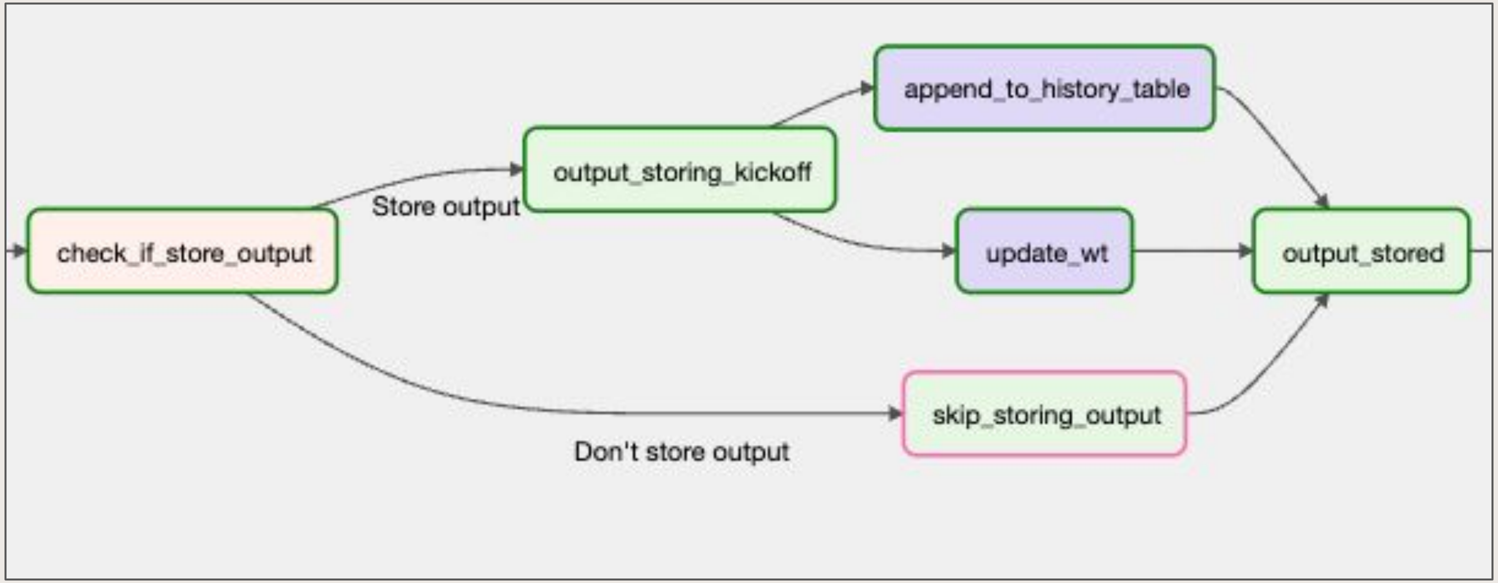
store_to_history_wt_and_events:

dags:

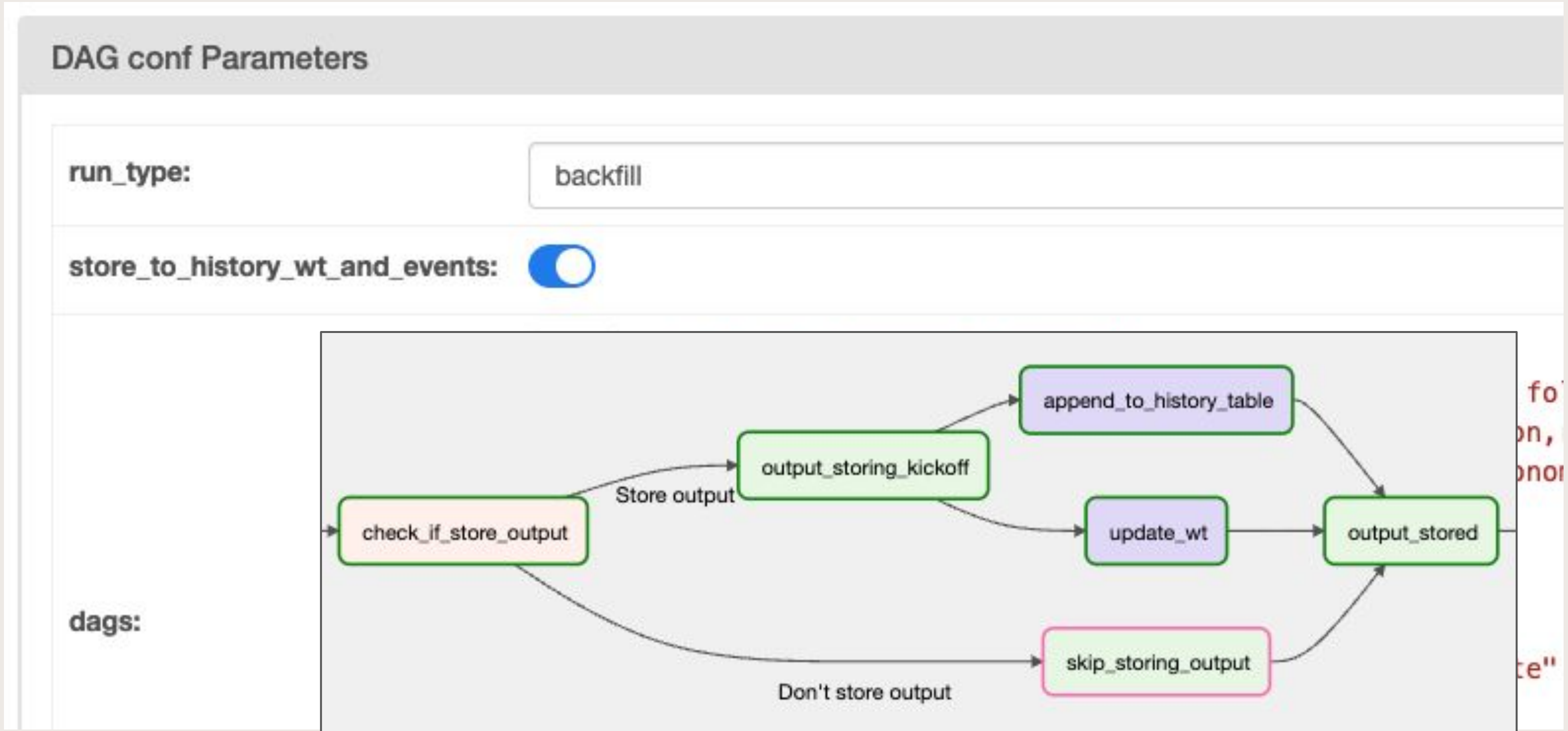
```
    "sc_non_business_taxonomy": {
        "essence": "Classifies the site as one of the fo
        "history_fields": "msid,execution_date,revision,
        "history_table": "prod.ds.sc_non_business_taxonor
        "input_dag": "sc_feature_enrichment",
        "model": "ds-sc-out-of-taxonomy",
        "short_name": "non_business_taxonomy",
        "wt": "prod.ds.sc_non_business_taxonomy_wt",
        "wt_date_column": "revision_date,execution_date"
    },
```

New Tasks: Store Results

```
ds_dag.new_tasks_store_results()
```



Ad-hoc runs

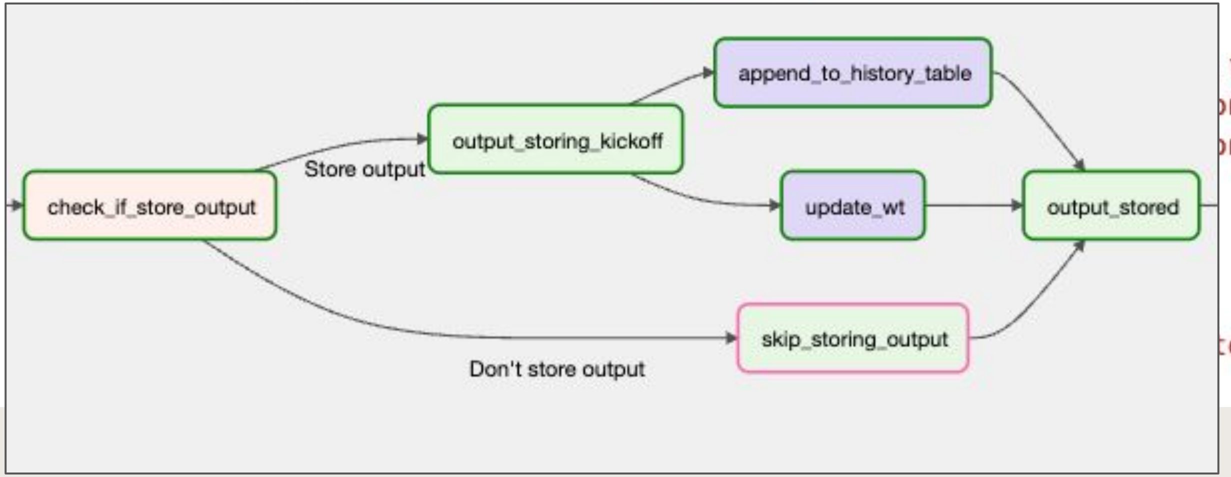


DAG conf Parameters

run_type: backfill

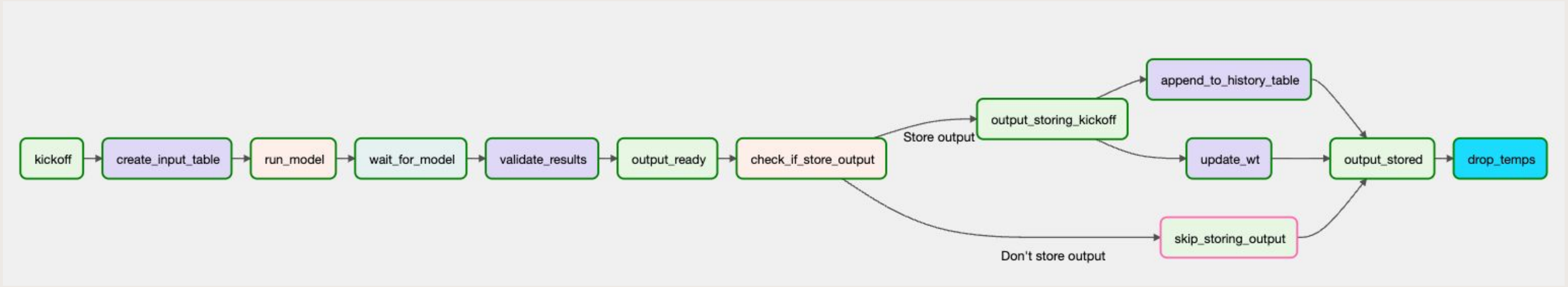
store_to_history_wt_and_events:

dags:

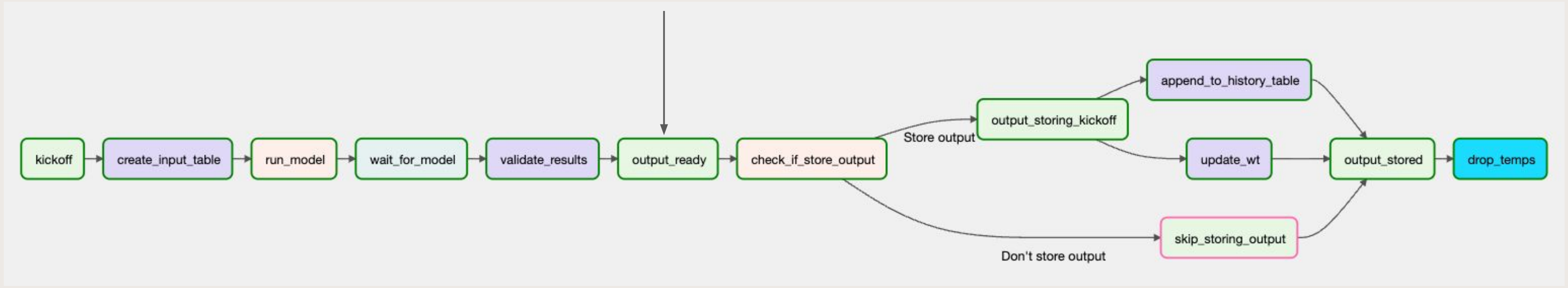


Problem 3:
Too Slow

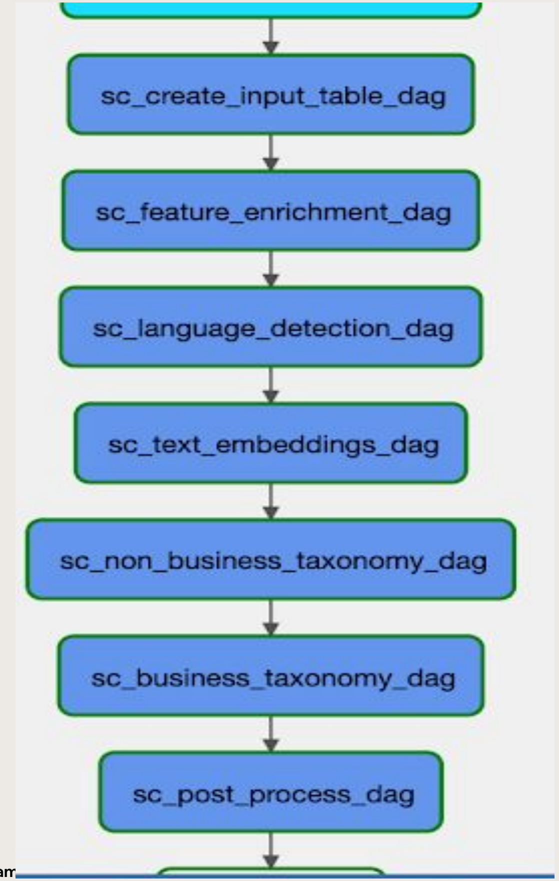
Internal DAG



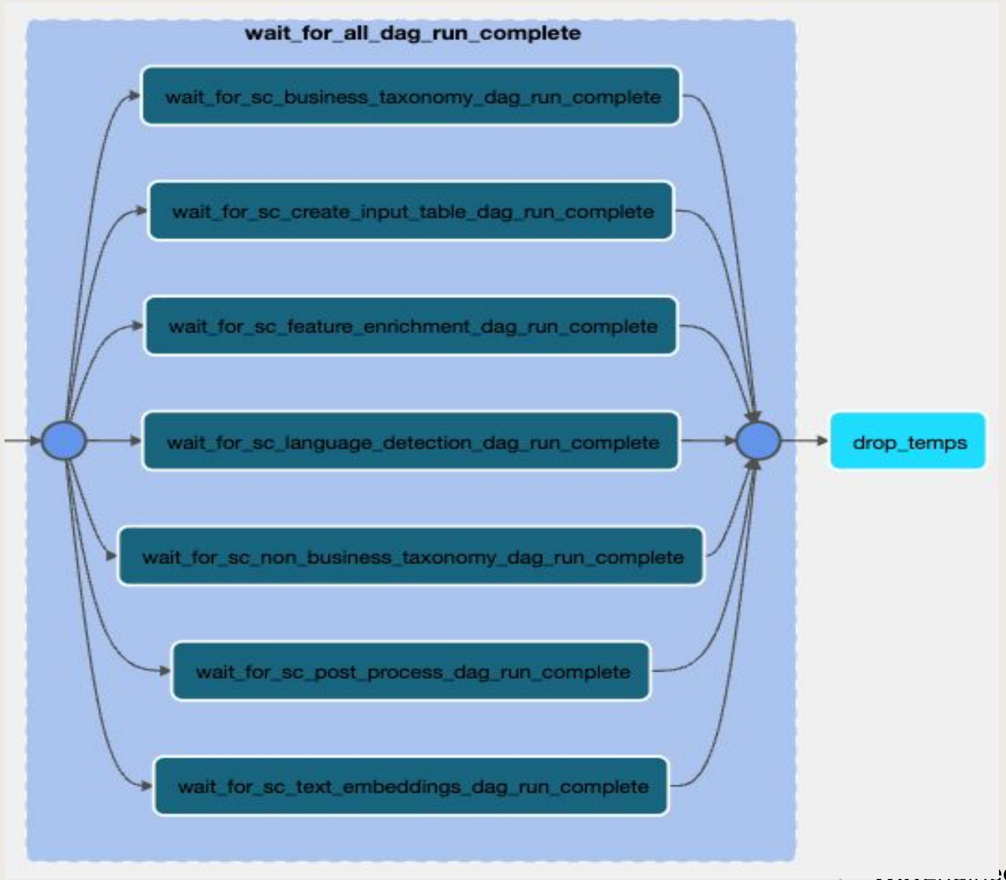
Wait for Output Ready



Part 1


sc_create_input_table_dag
sc_feature_enrichment_dag
sc_language_detection_dag
sc_text_embeddings_dag
sc_non_business_taxonomy_dag
sc_business_taxonomy_dag
sc_post_process_dag

Part 2


wait_for_all_dag_run_complete
wait_for_sc_business_taxonomy_dag_run_complete
wait_for_sc_create_input_table_dag_run_complete
wait_for_sc_feature_enrichment_dag_run_complete
wait_for_sc_language_detection_dag_run_complete
wait_for_sc_non_business_taxonomy_dag_run_complete
wait_for_sc_post_process_dag_run_complete
wait_for_sc_text_embeddings_dag_run_complete
drop_temps

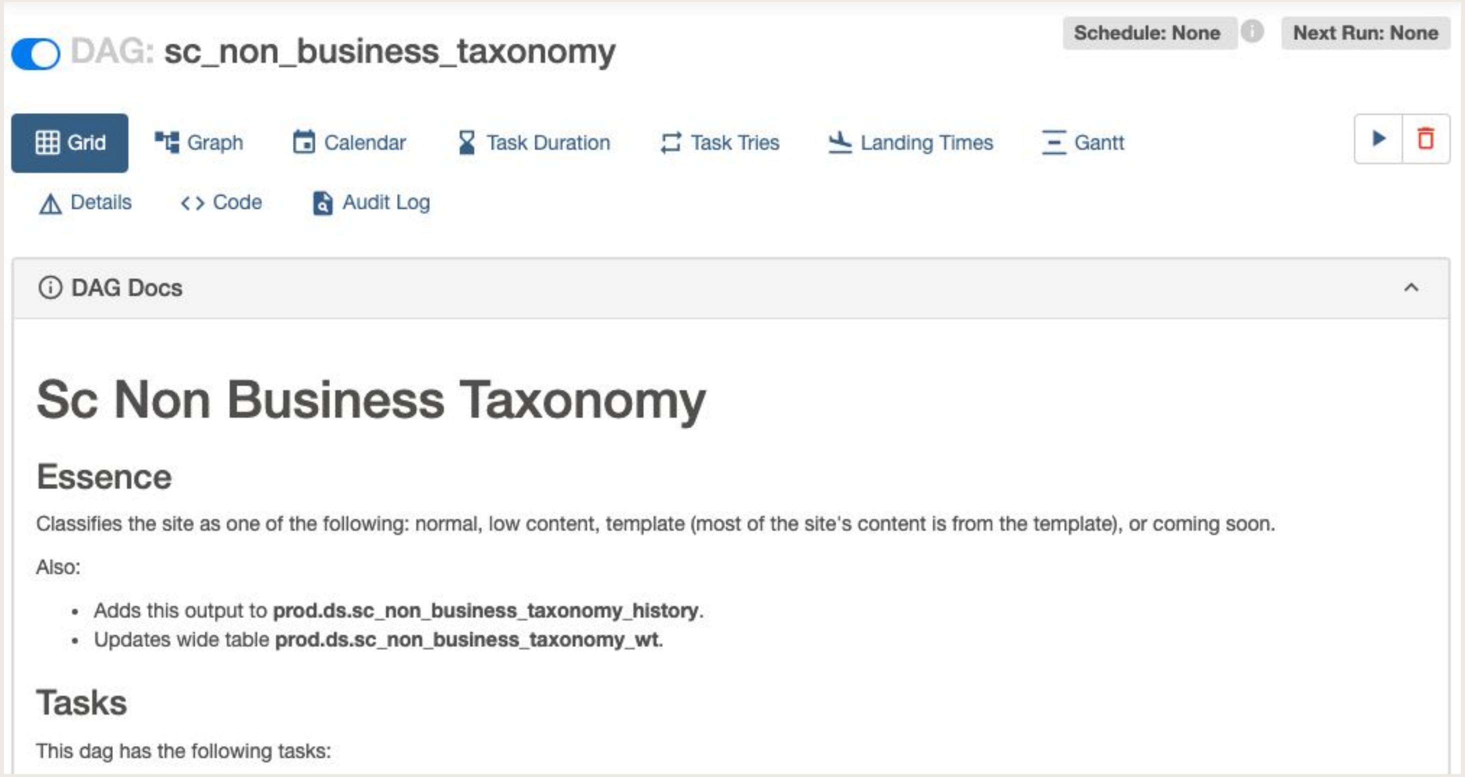
Problem 4:
Local Testing

local_config.py (for Site Classification)

```python
local_config = {
    'prefix': 'sandbox.gilr_',
    'sql_row_limit': 10_000
}
```

Problem 5: Documentation

Documentation



Sc Non Business Taxonomy

Essence

Classifies the site as one of the following: normal, low content, template (most of the site's content is from the template), or coming soon.

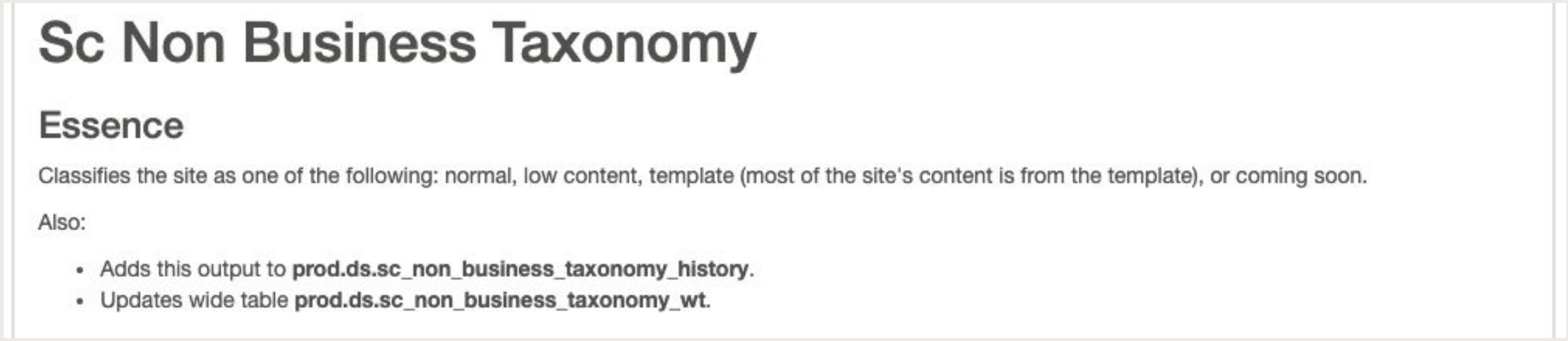
Also:

- Adds this output to **prod.ds.sc_non_business_taxonomy_history**.
- Updates wide table **prod.ds.sc_non_business_taxonomy_wt**.

Tasks

This dag has the following tasks:

```
# {{ title }}

### Essence

{{ essence }}
{{ more_info }}
```



Sc Non Business Taxonomy

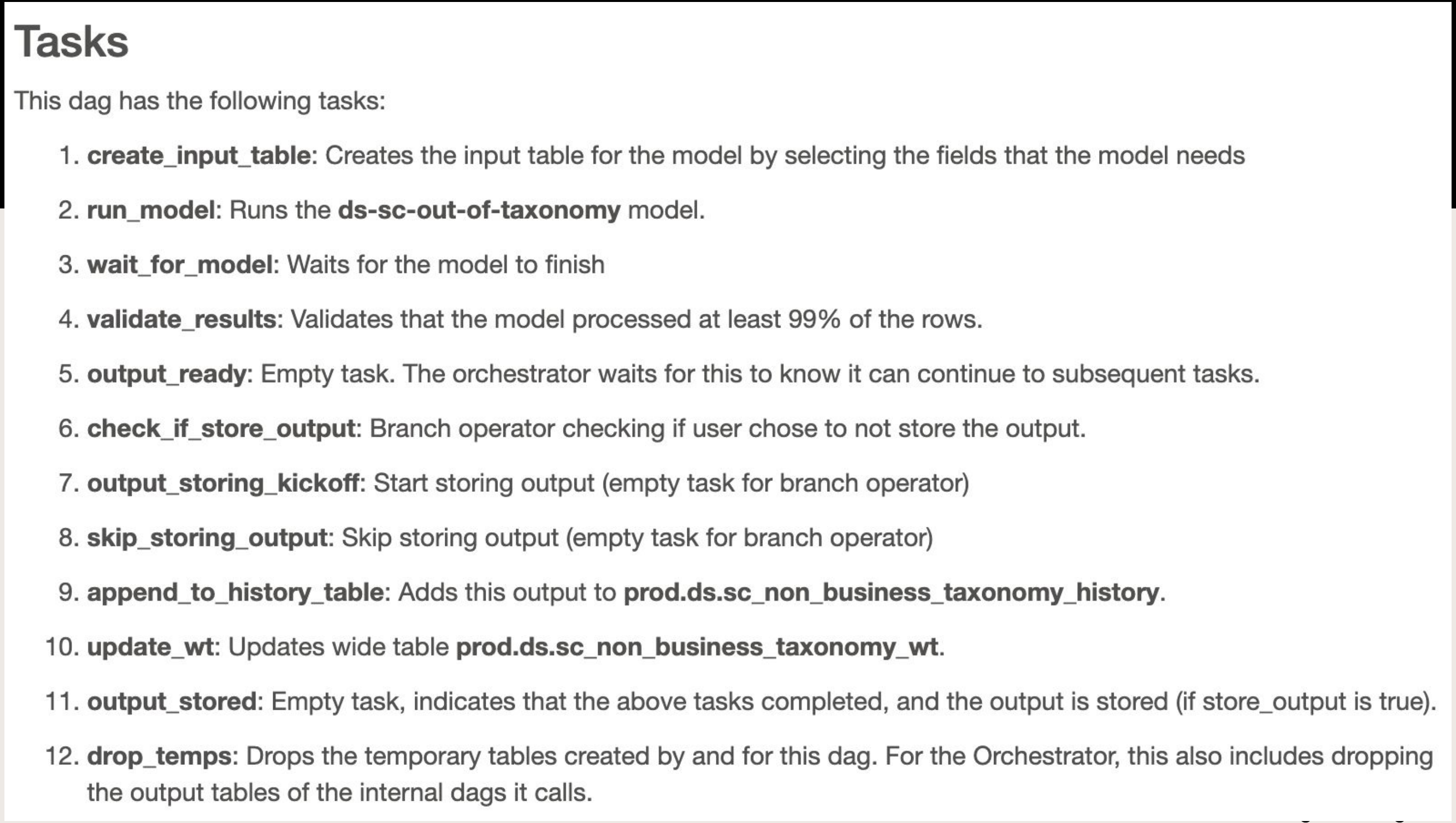
Essence

Classifies the site as one of the following: normal, low content, template (most of the site's content is from the template), or coming soon.

Also:

- Adds this output to **prod.ds.sc_non_business_taxonomy_history**.
- Updates wide table **prod.ds.sc_non_business_taxonomy_wt**.

```
### Tasks

This dag has the following tasks:


{{ tasks }}
```

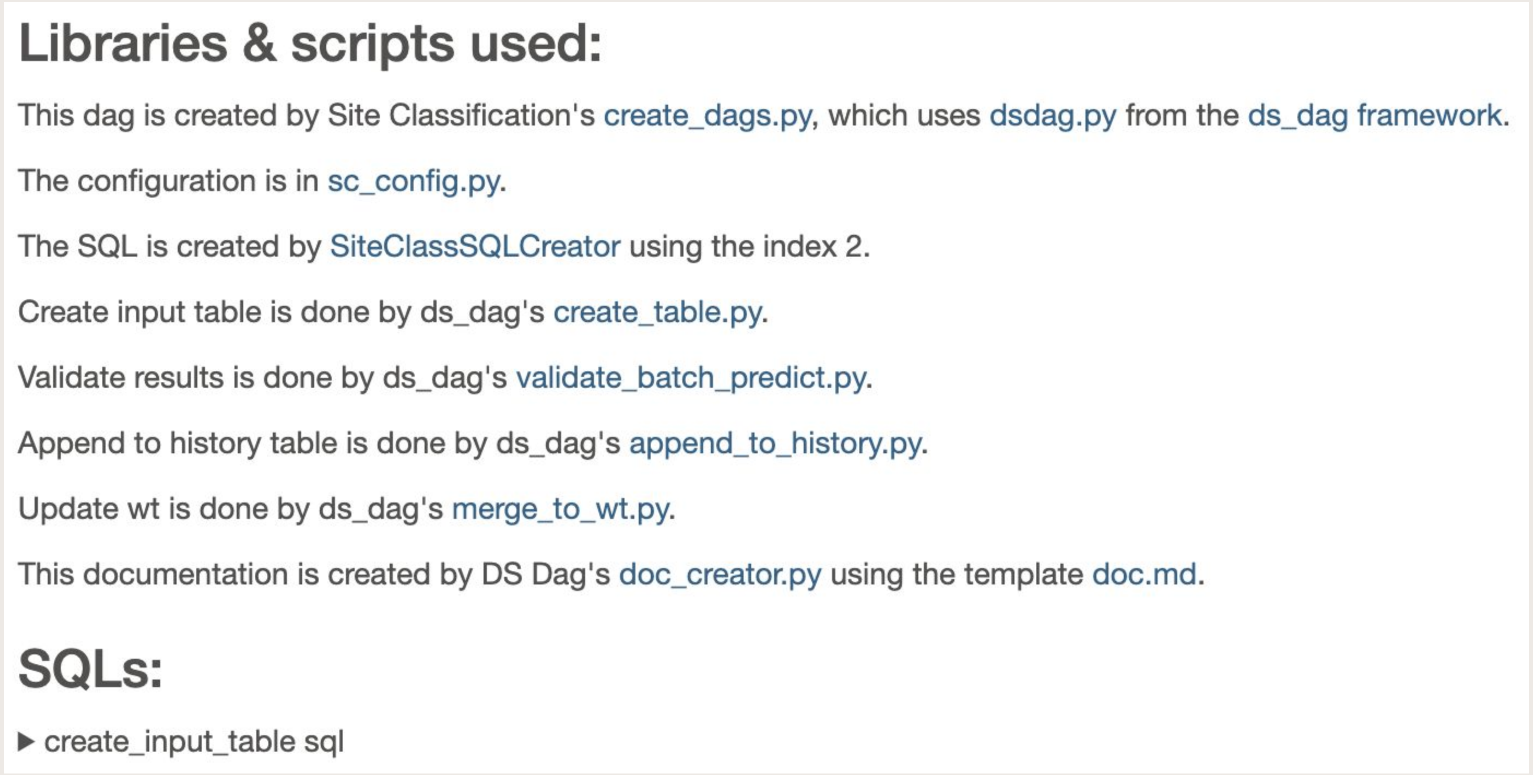


Tasks

This dag has the following tasks:

1. **create_input_table**: Creates the input table for the model by selecting the fields that the model needs
2. **run_model**: Runs the **ds-sc-out-of-taxonomy** model.
3. **wait_for_model**: Waits for the model to finish
4. **validate_results**: Validates that the model processed at least 99% of the rows.
5. **output_ready**: Empty task. The orchestrator waits for this to know it can continue to subsequent tasks.
6. **check_if_store_output**: Branch operator checking if user chose to not store the output.
7. **output_storing_kickoff**: Start storing output (empty task for branch operator)
8. **skip_storing_output**: Skip storing output (empty task for branch operator)
9. **append_to_history_table**: Adds this output to **prod.ds.sc_non_business_taxonomy_history**.
10. **update_wt**: Updates wide table **prod.ds.sc_non_business_taxonomy_wt**.
11. **output_stored**: Empty task, indicates that the above tasks completed, and the output is stored (if store_output is true).
12. **drop_temps**: Drops the temporary tables created by and for this dag. For the Orchestrator, this also includes dropping the output tables of the internal dags it calls.

Auto-Generated Documentation



Libraries & scripts used:

This dag is created by Site Classification's create_dags.py, which uses dsdag.py from the ds_dag framework.

The configuration is in sc_config.py.

The SQL is created by SiteClassSQLCreator using the index 2.

Create input table is done by ds_dag's create_table.py.

Validate results is done by ds_dag's validate_batch_predict.py.

Append to history table is done by ds_dag's append_to_history.py.

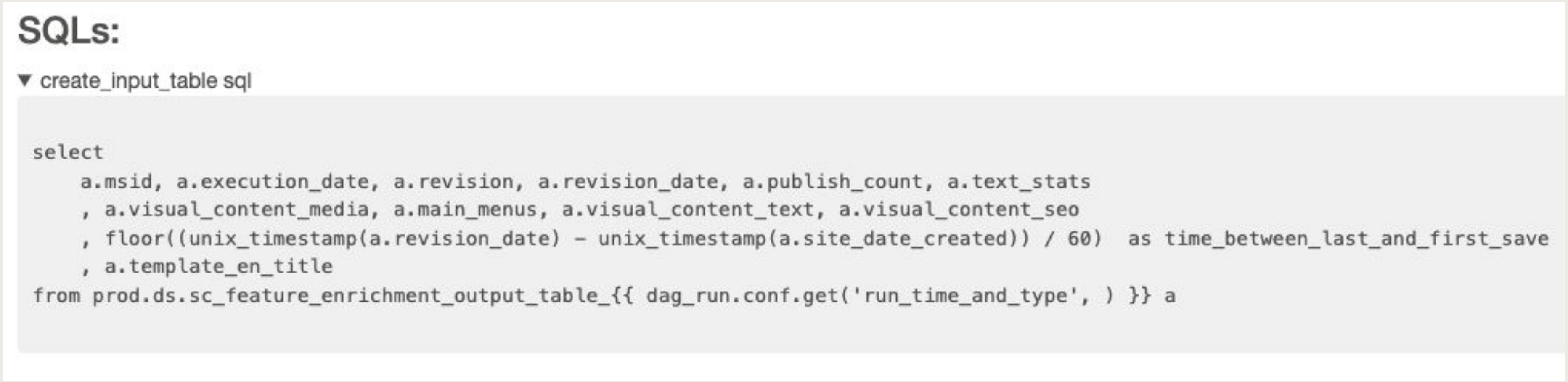
Update wt is done by ds_dag's merge_to_wt.py.

This documentation is created by DS Dag's doc_creator.py using the template doc.md.

SQLs:

▶ create_input_table sql

Auto-Generated Documentation

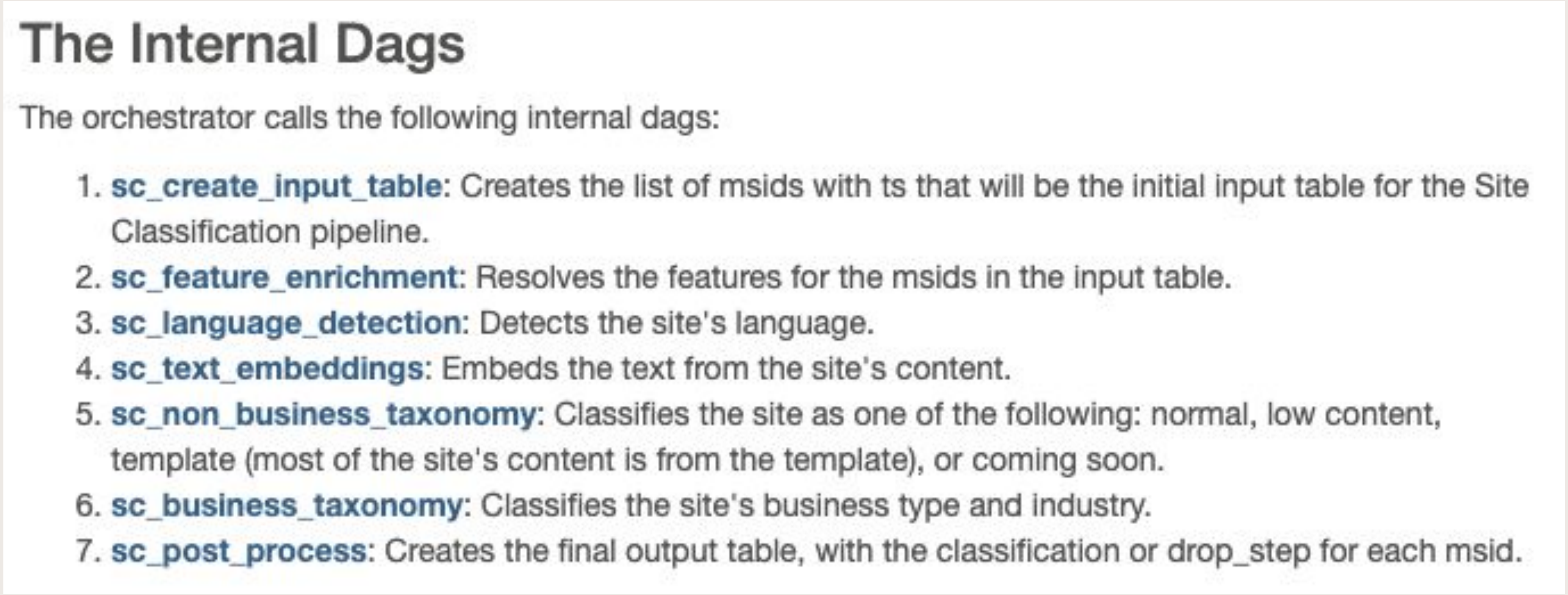


SQLs:

▼ create_input_table sql

```
select
    a.msid, a.execution_date, a.revision, a.revision_date, a.publish_count, a.text_stats
    , a.visual_content_media, a.main_menus, a.visual_content_text, a.visual_content_seo
    , floor((unix_timestamp(a.revision_date) - unix_timestamp(a.site_date_created)) / 60)  as time_between_last_and_first_save
    , a.template_en_title
from prod.ds.sc_feature_enrichment_output_table_{{ dag_run.conf.get('run_time_and_type', ) }} a
```

Auto-Generated Documentation: Orchestrators

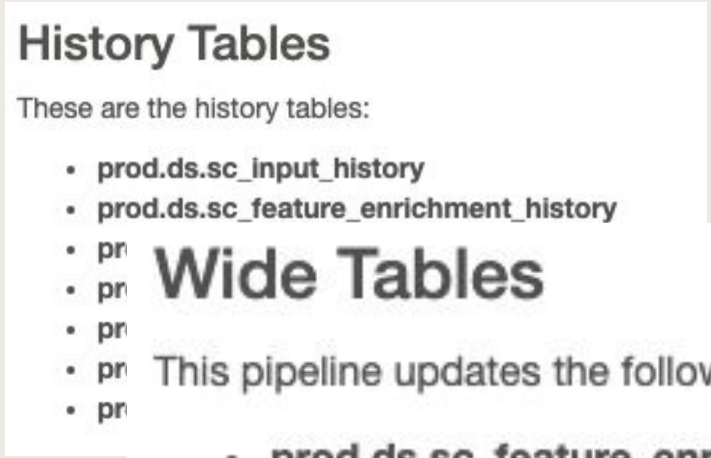


The Internal Dags

The orchestrator calls the following internal dags:

1. **sc_create_input_table**: Creates the list of msids with ts that will be the initial input table for the Site Classification pipeline.
2. **sc_feature_enrichment**: Resolves the features for the msids in the input table.
3. **sc_language_detection**: Detects the site's language.
4. **sc_text_embeddings**: Embeds the text from the site's content.
5. **sc_non_business_taxonomy**: Classifies the site as one of the following: normal, low content, template (most of the site's content is from the template), or coming soon.
6. **sc_business_taxonomy**: Classifies the site's business type and industry.
7. **sc_post_process**: Creates the final output table, with the classification or drop_step for each msid.

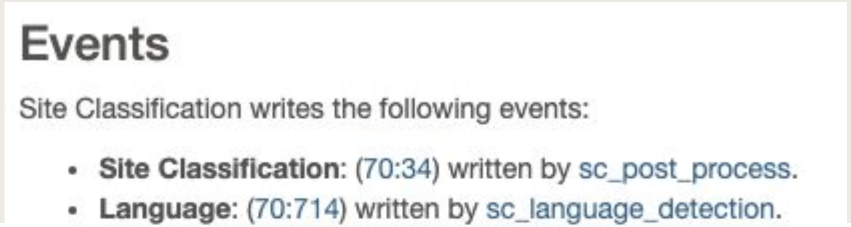
Auto-Generated Documentation: Orchestrators



History Tables

These are the history tables:

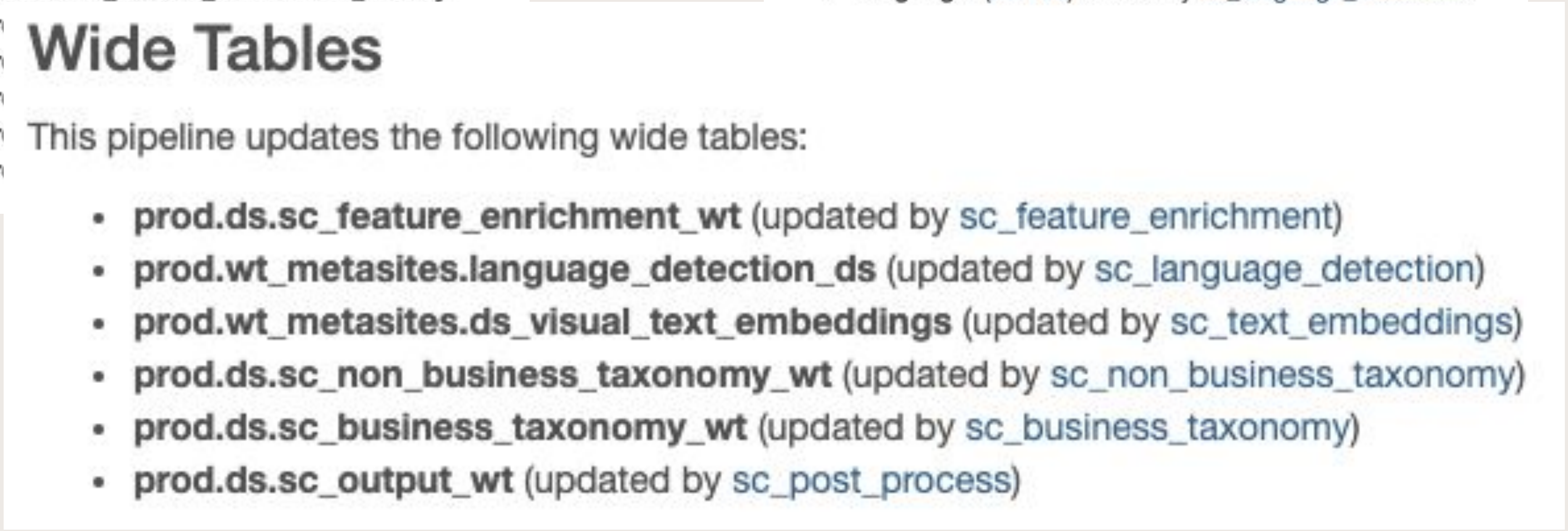
- **prod.ds.sc_input_history**
- **prod.ds.sc_feature_enrichment_history**
- **pr**
- **pr**
- **pr**
- **pr**
- **pr**



Events

Site Classification writes the following events:

- **Site Classification**: (70:34) written by sc_post_process.
- **Language**: (70:714) written by sc_language_detection.



Wide Tables

This pipeline updates the following wide tables:

- **prod.ds.sc_feature_enrichment_wt** (updated by sc_feature_enrichment)
- **prod.wt_metasites.language_detection_ds** (updated by sc_language_detection)
- **prod.wt_metasites.ds_visual_text_embeddings** (updated by sc_text_embeddings)
- **prod.ds.sc_non_business_taxonomy_wt** (updated by sc_non_business_taxonomy)
- **prod.ds.sc_business_taxonomy_wt** (updated by sc_business_taxonomy)
- **prod.ds.sc_output_wt** (updated by sc_post_process)

Problem 6:
Our other projects need this too

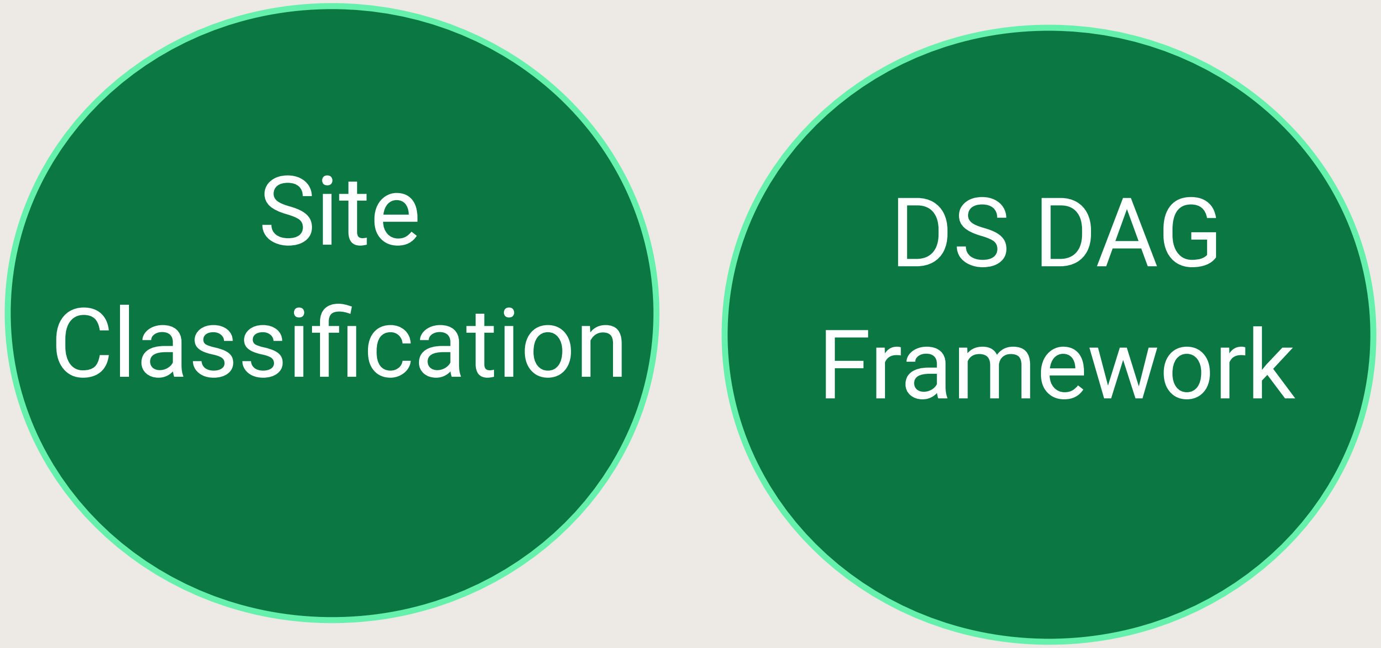
Wix & Airflow

1500
Airflow DAGs

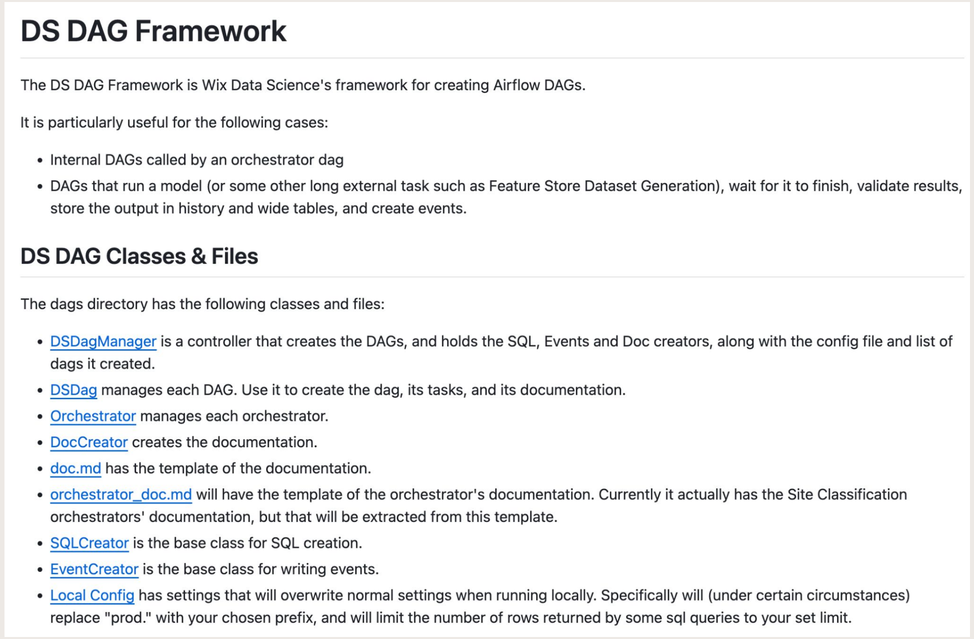
12K
Airflow tasks

5500
Daily DAG runs

DS DAG Framework



DS DAG Framework



DS DAG Framework

The DS DAG Framework is Wix Data Science's framework for creating Airflow DAGs.

It is particularly useful for the following cases:

- Internal DAGs called by an orchestrator dag
- DAGs that run a model (or some other long external task such as Feature Store Dataset Generation), wait for it to finish, validate results, store the output in history and wide tables, and create events.

DS DAG Classes & Files

The dags directory has the following classes and files:

- DSDagManager is a controller that creates the DAGs, and holds the SQL, Events and Doc creators, along with the config file and list of dags it created.
- DSDag manages each DAG. Use it to create the dag, its tasks, and its documentation.
- Orchestrator manages each orchestrator.
- DocCreator creates the documentation.
- doc.md has the template of the documentation.
- orchestrator_doc.md will have the template of the orchestrator's documentation. Currently it actually has the Site Classification orchestrators' documentation, but that will be extracted from this template.
- SQLCreator is the base class for SQL creation.
- EventCreator is the base class for writing events.
- Local Config has settings that will overwrite normal settings when running locally. Specifically will (under certain circumstances) replace "prod." with your chosen prefix, and will limit the number of rows returned by some sql queries to your set limit.

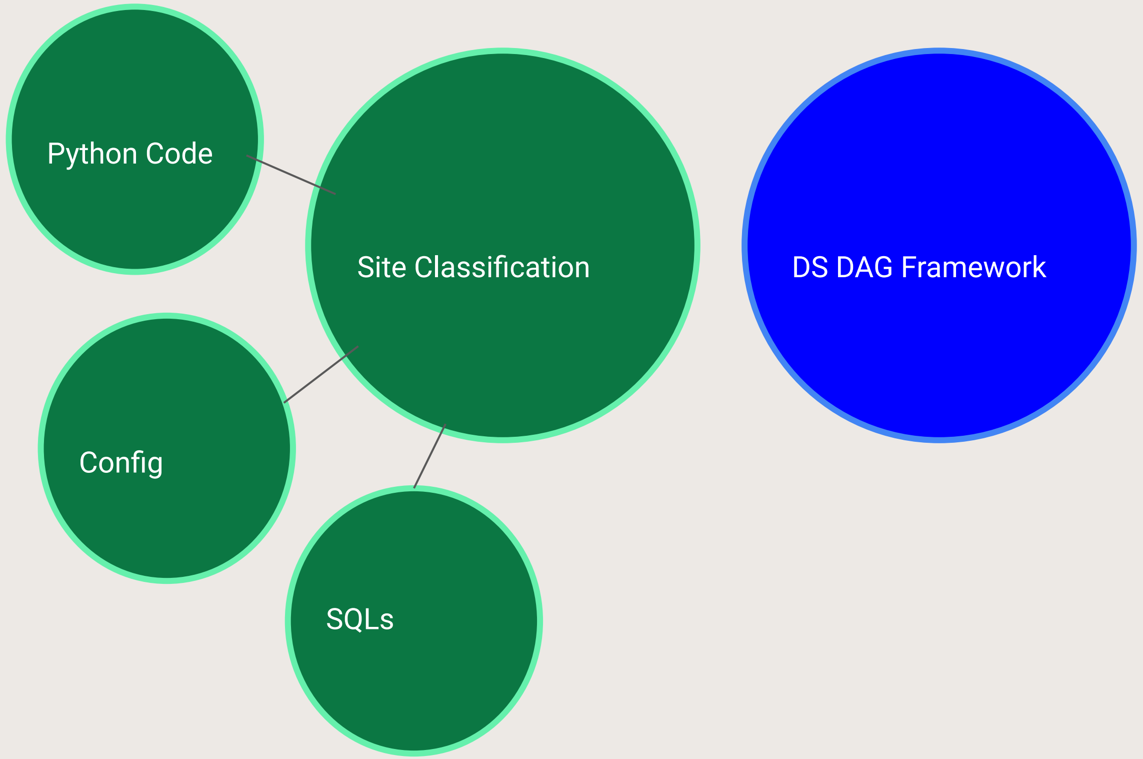
Code & Read Me: https://github.com/gil2/ds-dag-framework

Summary

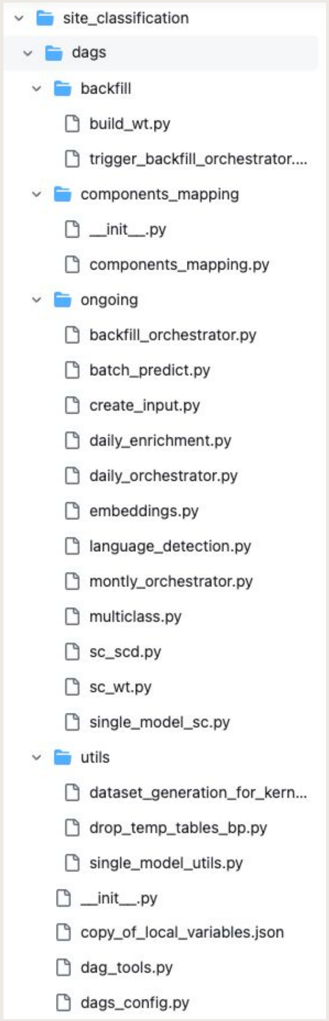
Problems we addressed:

1. Duplicate code
2. Ad-hoc runs
3. Local testing
4. Too slow
5. Documentation
6. Share this framework

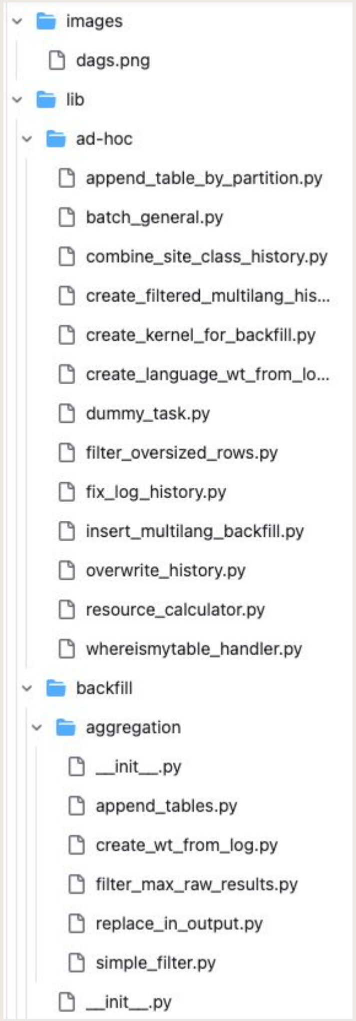
Site Classification & DS DAG Framework



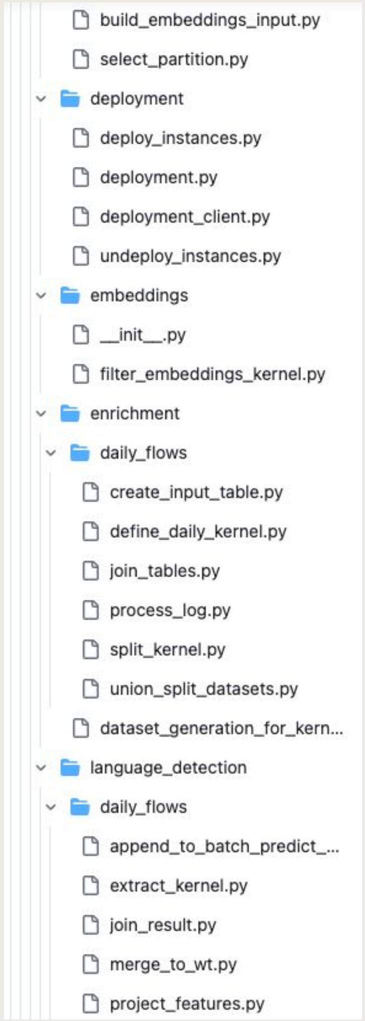
Before



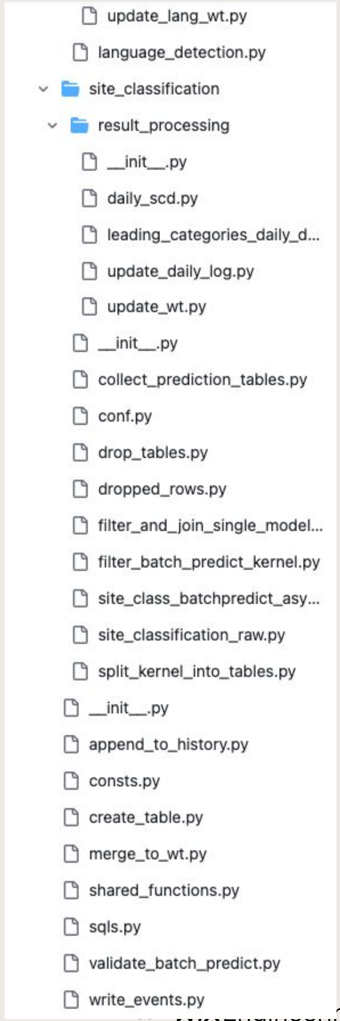
```
site_classification
  dags
    backfill
      build_wt.py
      trigger_backfill_orchestrator....
    components_mapping
      __init__.py
      components_mapping.py
    ongoing
      backfill_orchestrator.py
      batch_predict.py
      create_input.py
      daily_enrichment.py
      daily_orchestrator.py
      embeddings.py
      language_detection.py
      montly_orchestrator.py
      multiclass.py
      sc_scd.py
      sc_wt.py
      single_model_sc.py
    utils
      dataset_generation_for_kern...
      drop_temp_tables_bp.py
      single_model_utils.py
    __init__.py
    copy_of_local_variables.json
    dag_tools.py
    dags_config.py
```



```
images
  dags.png
lib
  ad-hoc
    append_table_by_partition.py
    batch_general.py
    combine_site_class_history.py
    create_filtered_multilang_his...
    create_kernel_for_backfill.py
    create_language_wt_from_lo...
    dummy_task.py
    filter_oversized_rows.py
    fix_log_history.py
    insert_multilang_backfill.py
    overwrite_history.py
    resource_calculator.py
    whereismytable_handler.py
  backfill
    aggregation
      __init__.py
      append_tables.py
      create_wt_from_log.py
      filter_max_raw_results.py
      replace_in_output.py
      simple_filter.py
    __init__.py
```

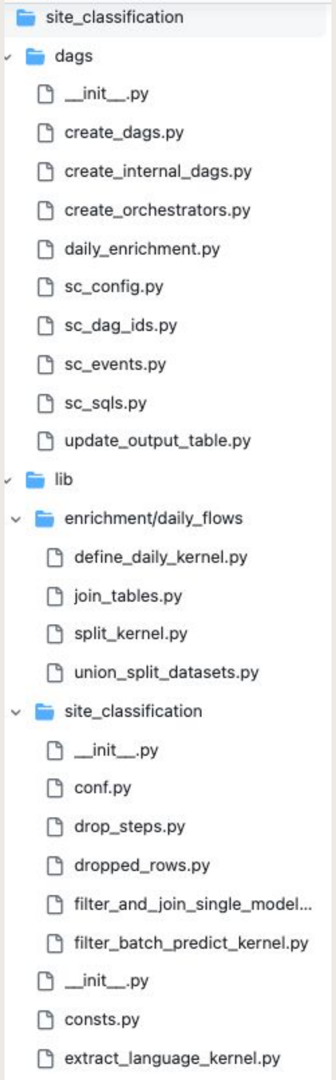


```
    build_embeddings_input.py
    select_partition.py
  deployment
    deploy_instances.py
    deployment.py
    deployment_client.py
    undeploy_instances.py
  embeddings
    __init__.py
    filter_embeddings_kernel.py
  enrichment
    daily_flows
      create_input_table.py
      define_daily_kernel.py
      join_tables.py
      process_log.py
      split_kernel.py
      union_split_datasets.py
    dataset_generation_for_kern...
  language_detection
    daily_flows
      append_to_batch_predict_...
      extract_kernel.py
      join_result.py
      merge_to_wt.py
      project_features.py
```

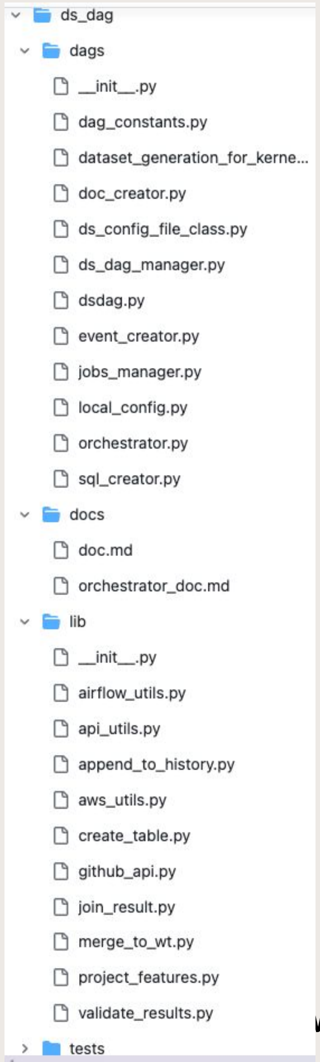


```
      update_lang_wt.py
    language_detection.py
  site_classification
    result_processing
      __init__.py
      daily_scd.py
      leading_categories_daily_d...
      update_daily_log.py
      update_wt.py
    __init__.py
    collect_prediction_tables.py
    conf.py
    drop_tables.py
    dropped_rows.py
    filter_and_join_single_model...
    filter_batch_predict_kernel.py
    site_class_batchpredict_asy...
    site_classification_raw.py
    split_kernel_into_tables.py
  __init__.py
  append_to_history.py
  consts.py
  create_table.py
  merge_to_wt.py
  shared_functions.py
  sqls.py
  validate_batch_predict.py
  write_events.py
```

After

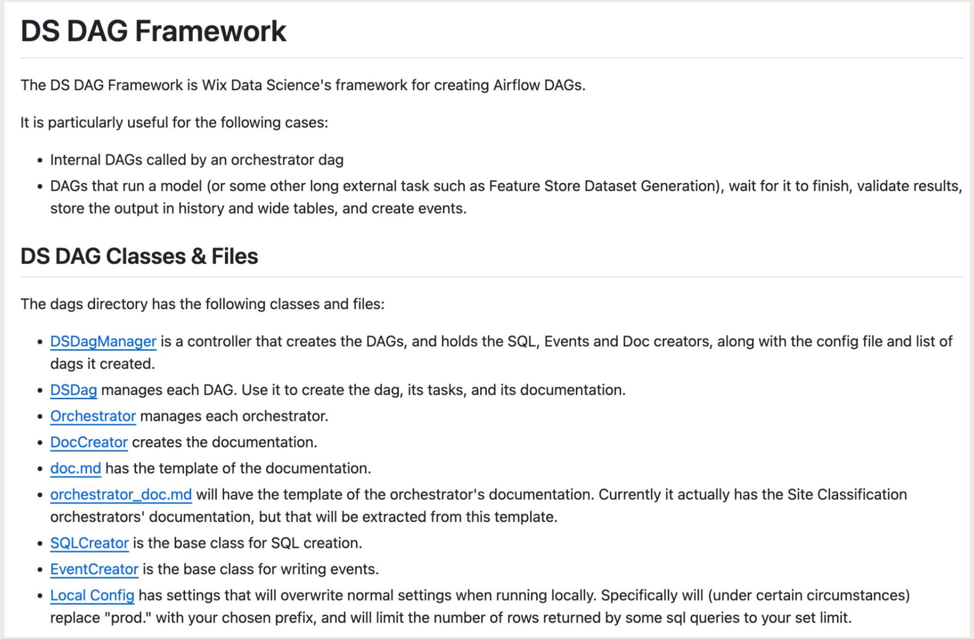


```
site_classification
  dags
    __init__.py
    create_dags.py
    create_internal_dags.py
    create_orchestrators.py
    daily_enrichment.py
    sc_config.py
    sc_dag_ids.py
    sc_events.py
    sc_sqls.py
    update_output_table.py
  lib
    enrichment/daily_flows
      define_daily_kernel.py
      join_tables.py
      split_kernel.py
      union_split_datasets.py
    site_classification
      __init__.py
      conf.py
      drop_steps.py
      dropped_rows.py
      filter_and_join_single_model...
      filter_batch_predict_kernel.py
    __init__.py
    consts.py
    extract_language_kernel.py
```



```
ds_dag
  dags
    __init__.py
    dag_constants.py
    dataset_generation_for_kerne...
    doc_creator.py
    ds_config_file_class.py
    ds_dag_manager.py
    dsdag.py
    event_creator.py
    jobs_manager.py
    local_config.py
    orchestrator.py
    sql_creator.py
  docs
    doc.md
    orchestrator_doc.md
  lib
    __init__.py
    airflow_utils.py
    api_utils.py
    append_to_history.py
    aws_utils.py
    create_table.py
    github_api.py
    join_result.py
    merge_to_wt.py
    project_features.py
    validate_results.py
  tests
```

DS DAG Framework



DS DAG Framework

The DS DAG Framework is Wix Data Science's framework for creating Airflow DAGs.

It is particularly useful for the following cases:

- Internal DAGs called by an orchestrator dag
- DAGs that run a model (or some other long external task such as Feature Store Dataset Generation), wait for it to finish, validate results, store the output in history and wide tables, and create events.

DS DAG Classes & Files

The dags directory has the following classes and files:

- DSDagManager is a controller that creates the DAGs, and holds the SQL, Events and Doc creators, along with the config file and list of dags it created.
- DSDag manages each DAG. Use it to create the dag, its tasks, and its documentation.
- Orchestrator manages each orchestrator.
- DocCreator creates the documentation.
- doc.md has the template of the documentation.
- orchestrator_doc.md will have the template of the orchestrator's documentation. Currently it actually has the Site Classification orchestrators' documentation, but that will be extracted from this template.
- SQLCreator is the base class for SQL creation.
- EventCreator is the base class for writing events.
- Local Config has settings that will overwrite normal settings when running locally. Specifically will (under certain circumstances) replace "prod." with your chosen prefix, and will limit the number of rows returned by some sql queries to your set limit.

Code & Read Me: https://github.com/gil2/ds-dag-framework

Thank you

Thank you!
Questions?

Code & Read Me:
https://github.com/gil2/ds-dag-framework

gilr@wix.com