HyperModel

Release 0.1.78

Nov 19, 2019

Contents:

CHAPTER 1

hypermodel package

1.1 Subpackages

1.1.1 hypermodel.cli package

1.1.1.1 Subpackages

hypermodel.cli.groups package

Submodules

hypermodel.cli.groups.k8s module

hypermodel.cli.groups.lake module

hypermodel.cli.groups.warehouse module

Module contents

1.1.1.2 Submodules

1.1.1.3 hypermodel.cli.cli_start module

```
hypermodel.cli.cli_start.main()
```

1.1.1.4 Module contents

1.1.2 hypermodel.features package

1.1.2.1 Submodules

1.1.2.2 hypermodel.features.categorical module

Helper functions for dealing with categorical features

hypermodel.features.categorical.**get_unique_feature_values**(*dataframe: pandas.core.frame.DataFrame, features: List[str]*) → Dict[str, List[str]]

Take a dataframe and a list of features, and for each feature find me all the unique

values of that feature. This is a useful step prior to one-hot encoding, as it gives you a list of all the values we can expect to encode.

Parameters

- **dataframe** (*pd.DataFrame*) – The DataFrame to use to collect values
- **features** (*List[str]*) – A list of all the Features we want to find the unique values of

Returns A dictionary keyed by the name of each feature, containing a list of all that features unique values

hypermodel.features.categorical.**one_hot_encode**(*dataframe: pandas.core.frame.DataFrame, uniques: Dict[str, List[str]], throw_on_missing=False*) → pandas.core.frame.DataFrame

Create a new dataframe that one-hot-encodes values from the given dataframe against the known list of unique feature values (calculated using *get_unique_feature_values*).

Parameters

- **dataframe** (*pd.DataFrame*) – The DataFrame to use to collect values
- **uniques** (*Dict[str, List[str]]*) – A dict keyed by feature name, containing a list of unique values
- **throw_on_missing** (*bool*) – If a value is found in the DataFrame which is missing from the *uniques* dict(), and this parameter is True, we will throw an Exception to prevent further execution. When encoding unseen data against known data, this can be useful to ensure you are not predicting using unseen data.

Returns A new DataFrame with each Feature/Value pair as a new column with a "1" where the row contains the features value, and a "0" where it does not

1.1.2.3 hypermodel.features.numerical module

hypermodel.features.numerical.**describe_features**(*dataframe: pandas.core.frame.DataFrame, features: List[str]*)

Return a dictionary keyed with the name of a feature and containing that features summary statistics.

Parameters

- **dataframe** (*pd.DataFrame*) – The dataframe to adjust values with
- **features** (*List[str]*) – The name of the features (columns in dataframe) to analyze

Returns A dictionary keyed by the feature name, containing summary statistics of the values of that feature.

hypermodel.features.numerical.**scale_by_mean_stdev**(*dataframe: pandas.core.frame.DataFrame*, *feature: str*, *mean: float*, *stdev: float*) → pandas.core.frame.DataFrame

Scale all the values in a column using a pre-sepcified mean / stdev, in place.

Parameters

- **dataframe** (*pd.DataFrame*) – The dataframe to adjust values with
- **feature** (*str*) – The name of the Feature column in the dataframe
- **mean** (*float*) – The mean to use to scale values
- **stdev** (*float*) – The standard deviation to use to scale values

Returns The adjusted dataframe passed in

1.1.2.4 Module contents

1.1.3 hypermodel.hml package

1.1.3.1 Subpackages

hypermodel.hml.prediction package

Module contents

1.1.3.2 Submodules

1.1.3.3 hypermodel.hml.decorators module

1.1.3.4 hypermodel.hml.hml_app module

1.1.3.5 hypermodel.hml.hml_container_op module

class hypermodel.hml.hml_container_op.**HmlContainerOp**(*func*, *kwargs*)

Bases: object

HmlContainerOp defines the base functionality for a Kubeflow Pipeline Operation which is executed as a simple command line application (assuming that the package) has been installed, and has a script based entry-point

invoke()

Actually invoke the function that this ContainerOp refers to (for testing / execution in the container)

Returns A reference to the current *HmlContainerOp* (self)

with_command(*container_command: str*, *container_args: List[str]*) → Optional[hypermodel.hml.hml_container_op.HmlContainerOp]

Set the command / arguments to execute within the container as a part of this job.

Parameters

- **container_command** (*str*) – The command to execute
- **container_args** (*List[str]*) – The arguments to pass the executable

Returns A reference to the current *HmlContainerOp* (self)

with_empty_dir(*name: str*, *mount_path: str*) → Optional[hypermodel.hml.hml_container_op.HmlContainerOp]
Create an empy, writable volume with the given *name* mounted to the specified *mount_path*

Parameters

- **name** (*str*) – The name of the volume to mount
- **mount_path** (*str*) – The path to mount the empty volume

Returns A reference to the current *HmlContainerOp* (self)

with_env(*variable_name*, *value*) → Optional[hypermodel.hml.hml_container_op.HmlContainerOp]
Bind an environment variable with the name *variable_name* and *value* specified

Parameters

- **variable_name** (*str*) – The name of the environment variable
- **value** (*str*) – The value to bind to the variable

Returns A reference to the current *HmlContainerOp* (self)

with_gcp_auth(*secret_name: str*) → Optional[hypermodel.hml.hml_container_op.HmlContainerOp]
Use the secret given in *secret_name* as the service account to use for GCP related SDK api calls (e.g. mount the secret to a path, then bind an environment variable GOOGLE_APPLICATION_CREDENTIALS to point to that path)

Parameters **secret_name** (*str*) – The name of the secret with the Google Service Account json file.

Returns A reference to the current *HmlContainerOp* (self)

with_image(*container_image_url: str*) → Optional[hypermodel.hml.hml_container_op.HmlContainerOp]
Set information about which container to use

Parameters

- **container_image_url** (*str*) – The url and tags for where we can find the container
- **container_command** (*str*) – The command to execute
- **container_args** (*List[str]*) – The arguments to pass the executable

Returns A reference to the current *HmlContainerOp* (self)

with_secret(*secret_name: str*, *mount_path: str*) → Optional[hypermodel.hml.hml_container_op.HmlContainerOp]
Bind a secret given by *secret_name* to the local path defined in *mount_path*

Parameters

- **secret_name** (*str*) – The name of the secret (in the same namespace)
- **mount_path** (*str*) – The path to mount the secret locally

Returns A reference to the current *HmlContainerOp* (self)

1.1.3.6 hypermodel.hml.hml_inference_app module

class hypermodel.hml.hml_inference_app.**HmlInferenceApp**(*name: str*, *cli: click.core.Group*, *image_url: str*, *package_entrypoint: str*, *port*, *k8s_namespace*)

Bases: object

The host of the Flask app used for predictions for models

apply_deployment(*k8s_deployment: kubernetes.client.models.extensions_v1beta1_deployment.ExtensionsV1beta1Deploym*

apply_service(*k8s_service: kubernetes.client.models.v1_service.V1Service*)

cli_inference_group = <click.core.Group object>

deploy()

get_model(*name: str*)

Get a reference to a model with the given name, retuning None if it cannot be found. :param name: The name of the model :type name: str

Returns The ModelContainer object of the model if it can be found, or None if it cannot be found.

on_deploy(*func: Callable[[hypermodel.hml.hml_inference_deployment.HmlInferenceDeployment], None]*)

on_init(*func: Callable*)

register_model(*model_container: hypermodel.hml.model_container.ModelContainer*)

Load the Model (its JobLib and Summary statistics) using an empy ModelContainer object, and bind it to our internal dictionary of models. :param model_container: The container wrapping the model :type model_container: ModelContainer

Returns The model container passed in, having been loaded.

start_dev()

Start the Flask App in development mode

start_prod()

Start the Flask App in Production mode (via Waitress)

1.1.3.7 hypermodel.hml.hml_inference_deployment module

class hypermodel.hml.hml_inference_deployment.**HmlInferenceDeployment**(*name: str*, *image_url: str*, *package_entrypoint: str*, *port*, *k8s_namespace*)

Bases: object

The *HmlInferenceDeployment* class provides functionality for managing deployments of the *HmlInferenceApp* to Kubernetes. This provides the ability to build and configure the required Kubernetes Deployments (Pods & Containers) along with a NodePort Service suitable for use with an Ingress (not created by this).

get_yaml()
Get the YAML like definition of the K8s Deployment and Service

with_empty_dir(*name: str*, *mount_path: str*) → Optional[hypermodel.hml.hml_inference_deployment.HmlInferenceDeplo
Create an empy, writable volume with the given *name* mounted to the specified *mount_path*

Parameters

- **name** (*str*) – The name of the volume to mount
- **mount_path** (*str*) – The path to mount the empty volume

Returns A reference to the current *HmlInferenceDeployment* (self)

with_env(*variable_name*, *value*) → Optional[hypermodel.hml.hml_inference_deployment.HmlInferenceDeployment]
Bind an environment variable with the name *variable_name* and *value* specified

Parameters

- **variable_name** (*str*) – The name of the environment variable
- **value** (*str*) – The value to bind to the variable

Returns A reference to the current *HmlInferenceDeployment* (self)

with_gcp_auth(*secret_name: str*) → Optional[hypermodel.hml.hml_inference_deployment.HmlInferenceDeployment]
Use the secret given in *secret_name* as the service account to use for GCP related SDK api calls (e.g. mount the secret to a path, then bind an environment variable GOOGLE_APPLICATION_CREDENTIALS to point to that path)

Parameters **secret_name** (*str*) – The name of the secret with the Google Service Account json file.

Returns A reference to the current *HmlInferenceDeployment* (self)

with_resources(*limit_cpu: str*, *limit_memory: str*, *request_cpu: str*, *request_memory: str*) → Optional[hypermodel.hml.hml_inference_deployment.HmlInferenceDeployment]
Set the Resource Limits and Requests for the Container running the *HmlInferenceApp*

Parameters

- **limit_cpu** (*str*) – Maximum amount of CPU to use
- **limit_memory** (*str*) – Maximum amount of Memory to use
- **request_cpu** (*str*) – The desired amount of CPU to reserve
- **request_memory** (*str*) – The desired amount of Memory to reserve

Returns A reference to the current *HmlInferenceDeployment* (self)

1.1.3.8 hypermodel.hml.hml_pipeline module

class hypermodel.hml.hml_pipeline.**HmlPipeline**(*cli: click.core.Group*, *pipeline_func: Callable*, *image_url: str*, *package_entrypoint: str*, *op_builders: List[Callable[[hypermodel.hml.hml_container_op.HmlContainerOp]]*)

Bases: object

apply_deploy_options(*func*)

Bind additional command line arguments for the deployment step, including: –host: Endpoint of the KFP API service to use –client-id: Client ID for IAP protected endpoint. –namespace: Kubernetes namespace to we want to deploy to

Parameters **func** (*Callable*) – The Click decorated function to bind options to

Returns The current *HmlPipeline* (self)

get_dag()
Get the calculated Argo Workflow Directed Acyclic Graph created by the Kubeflow Pipeline.ArithmeticError

Returns The "dag" object from the Argo workflow template.

run_all(***kwargs*)
Run all the steps in the pipeline

run_task(*task_name: str, run_log: Dict[str, bool], kwargs*)
Execute the Kubelow Operation for real, and mark the task as executed in the dict *run_log* so that we don't re-execute tasks that have already been executed.

Parameters

- **task_name** (*str*) – The name of the task/op to execute
- **run_log** (*Dict[str, bool]*) – A dictionary of all the tasks/ops we have already run
- **kwargs** – Additional keywork arguments to pass into the execution of the task

Returns None

with_cron(*cron: str*) → Optional[hypermodel.hml.hml_pipeline.HmlPipeline]
Bind a *cron* expression to the Pipeline, telling Kubeflow to execute the Pipeline on the specified schedule

Parameters **[str]** (*cron*) – The crontab expression to schedule execution

Returns The current *HmlPipeline* (self)

with_experiment(*experiment: str*) → Optional[hypermodel.hml.hml_pipeline.HmlPipeline]
Bind execution jobs to the specified experiment (only one).

Parameters **experiment** (*str*) – The name of the experiment

Returns The current *HmlPipeline* (self)

1.1.3.9 hypermodel.hml.hml_pipeline_app module

class hypermodel.hml.hml_pipeline_app.**HmlPipelineApp**(*name: str, cli: click.core.Group, image_url: str, package_entrypoint: str*)

Bases: object

on_deploy(*func: Callable[[hypermodel.hml.hml_container_op.HmlContainerOp], hypermodel.hml.hml_container_op.HmlContainerOp]*)
Registers a function to be called for each ContainerOp defined in the Pipeline to enable us to configure the Operations within the container with secrets, environment variables and whatever else may be required.

Parameters **func** (*Callable*) – The function (accepting a HmlContainerOp as its only parameter) which configure the supplied HmlContainerOp

register_pipeline(*pipeline_func*, *cron: str*, *experiment: str*)
Register a Kubeflow Pipeline (e.g. a function decorated with @hml.pipeline)

Parameters

- **pipeline_func** (*Callable*) – The function defining the pipline
- **cron** (*str*) – A cron expression for the default job executing this pipelines

- **experiment** (*str*) – The kubeflow experiment to deploy the job to

Returns Nonw

1.1.3.10 hypermodel.hml.model_container module

class hypermodel.hml.model_container.**ModelContainer**(*name: str, project_name: str, features_numeric: List[str], features_categorical: List[str], target: str, services: hypermodel.platform.abstract.services.PlatformServicesBase*)

Bases: object

The *ModelContainer* class provides a wrapper for a Machine Learning model, detailing information about Features (numeric & categorical), information about the distributions of feature columns and potentially a reference to the current version of the model's *.joblib* file.

analyze_distributions(*data_frame: pandas.core.frame.DataFrame*)
Given a dataframe, find all the unique values for categorical features and the distribution of all the numerical features and store them within this object.

Parameters **data_frame** (*pd.DataFrame*) – The dataframe to analyze

Returns A reference to self

bind_model(*model*)

build_training_matrix(*data_frame: pandas.core.frame.DataFrame*)
Convert the provided *data_frame* to a matrix after one-hot encoding all the categorical features, using the currently cached *feature_uniques*

Parameters **data_frame** (*pd.DataFrame*) – The pandas dataframe to encode

Returns A numpy array of the encoded data

create_merge_request(*reference*, *description='New models!'*)

dump_distributions()
Write information about the distributions of features to the local filesystem

Returns The path to the file that was written

dump_model()

dump_reference(*reference*)

get_bucket_path(*filename*)

get_local_path(*filename*)

load(*reference_file=None*)
Given the provided reference file, look up the location of the model in the DataLake and load it into memory. This will load the .joblib file, as well as any distributions / unique values associeated with this model reference

Parameters **reference_file** (*str*) – The path of the reference json file

Returns None

load_distributions(*file_path: str*)

load_model()

publish()
Publish the model (as a Joblib)

1.1.3.11 Module contents

1.1.4 hypermodel.kubeflow package

1.1.4.1 Submodules

1.1.4.2 hypermodel.kubeflow.deploy module

Helper function to deploy and run a pipeline to a production environment, deploying the pipeline as a part of the "Production" experiment.

hypermodel.kubeflow.deploy.**deploy_pipeline**(*pipeline*, *environment: str = 'dev'*, *host: Optional[str] = None*, *client_id: Optional[str] = None*, *namespace: Optional[str] = None*)
Deploy the current pipeline Kubeflew in the provided `namespace` on the using the Kubeflow api found at `host` and authenticate using `client_id`.

Parameters

- **environment** (*str*) – The environment to create the pipelien in (e.g. "dev", "prod")
- **host** (*str*) – The host we can find the Kubeflow API at (e.g. https://{APP_NAME}.endpoints.{PROJECT_ID}.cloud.goog/pipeline)
- **client_id** (*str*) – The IAP client id we can use for authorisate (e.g. "XXXXXX-XXXXXXXXX.apps.googleusercontent.com")
- **namespace** (*str*) – The Kuberenetes / Kubeflow namespace to deploy to (e.g. kubeflow)

1.1.4.3 hypermodel.kubeflow.deploy_dev module

Helper function to deploy and run a pipeline to a development environment

hypermodel.kubeflow.deploy_dev.**deploy_to_dev**(*pipeline*)
Deploy the Kubeflow Pipelines Pipeline (e.g. a method decorated with *@dsl.pipeline*) to Kubeflow and execute it.

Parameters **pipeline** (*func*) – The *@dsl.pipeline* method describing the pipeline

Returns True if the deployment suceeds

1.1.4.4 hypermodel.kubeflow.kubeflow_client module

class hypermodel.kubeflow.kubeflow_client.**KubeflowClient**(*host: Optional[str] = None*, *client_id: Optional[str] = None*, *namespace: Optional[str] = 'kubeflow'*)
Bases: `object`

A wrapper of the existing Kubeflow Pipelines Client which enriches it to be able to access more of the Kubeflow Pipelines API.

create_experiment(*experiment_name*)
Create a new Kubeflow Pipelines Experiment (grouping of pipeliens / runs)

Parameters **experiment_name** (*str*) – The name of the experiment

Returns The Kubeflow experiement object created

create_job(*name: str*, *pipeline*, *experiment*, *description=None*, *enabled=True*, *max_concurrency=1*, *cron=None*)
Create a new Kubeflow Pipelines Job

Parameters

- **name** (*str*) – The name of the *Job*
- **pipeline** (*Pipeline*) – The *Pipeline* object to execute when the *Job* is called
- **experiment** (*Experiment*) – The *Experiment* object to create the *Job* in.
- **description** (*str*) – A description of what the *Job* is all about
- **enabled** (*bool*) – Should be *Job* be enabled?
- **max_concurrency** (*int*) – How many concurrent executions of the *Job* are allowed?
- **cron** (*str*) – The CRON expression to use to execute the job periodicalls

Returns The Kubeflow API response object.

create_pipeline(*pipeline_func*, *pipeline_name*)
Create a new Kubeflow Pipeline using the provided pipeline function

Parameters **pipeline_func** – The method decorated by @dsl.pipeline which defines the pipeline

Returns The Kubeflow Pipeline object created

delete_job(*job*)
Delete a *Job* using its job.id

Parameters **job** (*KubeflowJob*) – A *Job* object to delete

Returns True if the *Job* was deleted succesfully

delete_pipeline(*pipeline*)
Delete the specified *Pipeline*

Parameters **pipeline** – The pipeline object to delete

Returns True if successfull

find_experiment(*id=None*, *name=None*)
Look up an *Experiment* by its name or id. Returns None if the *Experiment* cannot be found. Both *id* and *name* are optional, but atleast one must be provided. Where both a provided, the function will return with the first *Experiment* matching either id or name.

Parameters

- **id** (*str*) – The *id* of the *Experiment* to find
- **name** (*string*) – The *name* of the *Experiment* to find

Returns A reference to the *Experiment* if found, and None if not.

find_job(*job_name*)
Look up a job by its name (in the current namespace). Returns None if the job cannot be found

Parameters **job_name** (*str*) – The name of the job to find

Returns A reference to the job if found, and None if not.

find_pipeline(*name*)
Look up a *Pipeline* by its name (in the current namespace). Returns None if the *Pipeline* cannot be found

Parameters **name** (*str*) – The name of the *Pipeline* to find

Returns A reference to the *Pipeline* if found, and *None* if not.

list_experiments()
List the Experiments in the current namespace

Returns A list of all the Experiments

list_jobs()
List the Jobs in the current namespace

Returns A list of all the Jobs

list_pipelines()
List the *Pipelines* in the current namespace

Returns A list of all the *Pipelines* in the current *Experiment*

list_runs(*experiment_name*)
List the *Runs* in the specified Exper'iment

Parameters **experiment_name** (*str*) – The name of the *Experiment*

Returns A list of all the *Runs* in the current *Experiment*

1.1.4.5 Module contents

1.1.5 hypermodel.model package

1.1.5.1 Submodules

1.1.5.2 hypermodel.model.table_schema module

class hypermodel.model.table_schema.**SqlColumn**(*column_name: str*, *column_type: str*, *nullable: bool*)

Bases: object

A simple class to model a Column in a Table within a DataWarehouse or DataMart

to_sql() → str
Generate an SQL snippet for the definition of this column.

Returns An SQL string with the columns definition, suitable for including in a Create Table script

class hypermodel.model.table_schema.**SqlTable**(*dataset_id: str*, *table_id: str*, *columns: List[hypermodel.model.table_schema.SqlColumn] = []*)

Bases: object

A simple class to model a Column in a Table within a DataWarehouse or DataMart

to_sql() → str
Generate a "CREATE TABLE" script for the table defined in this object

Returns An SQL string with the create table script

1.1.5.3 Module contents

1.1.6 hypermodel.platform package

1.1.6.1 Subpackages

hypermodel.platform.gcp package

Submodules

hypermodel.platform.gcp.config module

class hypermodel.platform.gcp.config.**GooglePlatformConfig**
Bases: hypermodel.platform.abstract.platform_config.PlatformConfig

hypermodel.platform.gcp.data_lake module

class hypermodel.platform.gcp.data_lake.**DataLake**(*config: hypermodel.platform.gcp.config.GooglePlatformConfig*)
Bases: hypermodel.platform.abstract.data_lake.DataLakeBase

download(*bucket_path: str*, *destination_local_path: str*, *bucket_name: str = None*) → bool

upload(*bucket_path: str*, *local_path: str*, *bucket_name: str = None*) → bool

hypermodel.platform.gcp.data_warehouse module

class hypermodel.platform.gcp.data_warehouse.**DataWarehouse**(*config: hypermodel.platform.gcp.config.GooglePlatformCon*
Bases: hypermodel.platform.abstract.data_warehouse.DataWarehouseBase

dataframe_from_query(*query: str*) → pandas.core.frame.DataFrame

dataframe_from_table(*dataset: str*, *table: str*) → pandas.core.frame.DataFrame

dry_run(*query: str*) → List[hypermodel.model.table_schema.SqlColumn]

import_csv(*bucket_path: str*, *dataset: str*, *table: str*) → bool

select_into(*query: str*, *output_dataset: str*, *output_table: str*) → bool

table_schema(*dataset: str*, *table: str*) → hypermodel.model.table_schema.SqlTable

hypermodel.platform.gcp.gcp_base_op module

class hypermodel.platform.gcp.gcp_base_op.**GcpBaseOp**(*config: hypermodel.platform.gcp.config.GooglePlatformConfig*, *pipeline_name: str*, *op_name: str*)
Bases: object

GcpBaseOp defines the base functionality for a Kubeflow Pipeline Operation providing a convenient wrapper over Kubeflow's ContainerOp for use within the Google Kubernetes Engine (GKE) on Google Cloud Platform

bind_env (*variable_name: str*, *value: str*)
Create an environment variable for the container with the given value

Parameters

- **variable_name** (*str*) – The name of the variable in the container
- **value** (*str*) – The value to bind to the variable

Returns A reference to the current GcpBaseOp (for chaining)

bind_gcp_auth (*gcp_auth_secret: str*)
Bind the `gcp_auth_secret` that contains the Service Account that this container should use to authenticate and authorise itself.

Parameters **gcp_auth_secret** (*str*) – The name of the secret containing the service account this container should use

Returns A reference to the current GcpBaseOp (for chaining)

bind_output_artifact_path (*name: str*, *path: str*)
Add an artifact to the Kubeflow Pipeline Operation using the `name` provided with the content from the `path` provided

Parameters

- **name** (*str*) – The name of the output artifact
- **path** (*str*) – The path to find the content for the artifact

Returns A reference to the current GcpBaseOp (for chaining)

bind_output_file_path (*name*, *path*)
Add an output file to the Kubeflow Pipeline Operation using the `name` provided with the content from the `path` provided

Parameters

- **name** (*str*) – The name of the output file
- **path** (*str*) – The path to find the content for the file

Returns A reference to the current GcpBaseOp (for chaining)

bind_secret (*secret_name: str*, *mount_path: str*)
Bind a secret with the name `secret_name` from Kubernetes (in the same namespace as the container) to the specified `mount_path`

Parameters

- **secret_name** (*str*) – The name of the secret to mount
- **mount_path** (*str*) – The path to mount the secret to

Returns A reference to the current GcpBaseOp (for chaining)

get (*key: str*)
Get the value of a variable bound to this Operation, returning None if the `key` is not found.

Parameters **key** (*str*) – The key to get the value of

Returns The value of the given `key`, or None if the key is not found in currently bound values.

op (*overrides={}*)
Generate a ContainerOp object from all the configuration stored as a part of this Op.

Parameters **overrides** (*Dict[str,str]*) – Override the bound variables with these values

Returns ContainerOp using settins from this op

with_container(*container_image_url: str, container_command: str, container_args: List[str]*)
Set information about which container to use, and the command in that container to execute as a part of this job.

Parameters

- **container_image_url** (*str*) – The url and tags for where we can find the container
- **container_command** (*str*) – The command to execute
- **container_args** (*List[str]*) – The arguments to pass the executable

hypermodel.platform.gcp.services module

class hypermodel.platform.gcp.services.**GooglePlatformServices**
Bases: hypermodel.platform.abstract.services.PlatformServicesBase

Services related to our Google Platform / Gitlab technology stack, including:

config
An object containing configuration information

Type *GooglePlatformConfig*

lake
A reference to DataLake functionality, implemented through Google Cloud Storage

Type *DataLake*

warehouse
A reference to DataWarehouse functionality implemented through BigQuery

Type *DataWarehouse*

config

git

lake

warehouse

Module contents

hypermodel.platform.local package

Submodules

hypermodel.platform.local.config module

hypermodel.platform.local.data_lake module

hypermodel.platform.local.data_warehouse module

hypermodel.platform.local.services module

Module contents

1.1.6.2 Module contents

1.1.7 hypermodel.utilities package

1.1.7.1 Submodules

1.1.7.2 hypermodel.utilities.file_hash module

`hypermodel.utilities.file_hash.`**`file_md5`**(*fname*)

1.1.7.3 hypermodel.utilities.hm_shell module

`hypermodel.utilities.hm_shell.`**`sh`**(*cmd: str*, *cwd: str = '.'*, *env=None*, *debug: bool = False*, *ignore_error: bool = False*) → Tuple[int, str, str]

Execures a shell command using 'subprocess.Popen', returning a tuple

1.1.7.4 hypermodel.utilities.k8s module

Utility functions to make it easier to work with Kubernetes, primarily just a wrapper around kubectl commands

`hypermodel.utilities.k8s.`**`connect`**(*cluster_name: str*, *zone: str*, *project: str*) → None

Using gcloud, set up the environment to connect to the specified cluster, given by `cluster_name` in the `zone` and `project`.

Parameters

- **`cluster_name`** (*str*) – The name of the cluster
- **`zone`** (*str*) – The zone the cluster was created in (e.g. 'australia-southeast1-a')
- **`project`** (*str*) – The google cloud project you wish to connect to

Returns Returns True if everything worked as expected

`hypermodel.utilities.k8s.`**`sanitize_k8s_name`**(*name: str*)

From _make_kubernetes_name sanitize_k8s_name cleans and converts the names in the workflow.

`hypermodel.utilities.k8s.`**`secret_from_env`**(*env_var: str*, *namespace: str*) → bool

Create a Kubernetes secret in the provided `namespace` using an environment variable given by `env_var`.

Parameters

- **`env_var`** – The name of the environment variable to save as a secret
- **`namespace`** – The Kubernetes namespace to save the secret in

Returns Returns True if everything worked as expected

`hypermodel.utilities.k8s.`**`secret_to_file`**(*secret_name: str*, *namespace: str*, *path: str*) → bool

Find the secret named `secret_name` in the namespace `namespace` and save it to a file at the path given by `path`

Parameters

- **secret_name** – The name of the secret we want to export
- **namespace** – The namespace that the secret lives in
- **path** – The path to a directory where we want to save the secret files

Returns Returns True if everything worked as expected

1.1.7.5 hypermodel.utilities.kubeflow module

Utility functions for working with Kubeflow

hypermodel.utilities.kubeflow.**am_in_kubeflow**() → bool
Answers the question: 'Am I currently being executed in a Kubeflow Pipeline Workflow by checking to see if we have an environment variables called 'KF_WORKFLOW_ID'

Returns True if I am running in a Kubeflow Pipelines

1.1.7.6 Module contents

1.2 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

F

G

H

I

K

L

M

O

P

R

S

T

U

W