Darwin™ Python SDK Manual

A SparkCognition™ Education Document

v. 1.0 - 02.05.2018

Contents

This document contains the following sections:

About this guide

This manual describes using the Darwin™ SDK to access and use the Darwin API in automated model building. It is intended for data scientists, software engineers and analysts who want to use the Darwin API to interact with Darwin to create and train models, monitor jobs and perform analysis.The SDK also provides some convenience functions. Note that throughout this document, long key and token values are truncated, indicated by ellipses (**…**).

Expectation

This document assumes experience of the data scientist or software engineer that is commensurate with data science techniques and associated programming tasks.

Darwin overview

Darwin is a SparkCognition® tool that automates model building processes to solve specific problems. This tool enhances data scientist potential because it automates various tasks that are often manually performed. These tasks include data cleaning, latent relationship extraction, and optimal model determination. Darwin promotes rapid and accurate feature generation through both automated windowing and risk generation. Darwin quickly creates highly-accurate, dynamic models using both supervised and unsupervised learning methods.
The general workflow for simple modeling includes:

- Upload training data
- Create model
- Upload test data

- Test the model
- Download result artifact

Note: Darwin expects all uploaded ingestion files to be in a *rectangular* format. This means a flat file with features that span columns and data samples that span rows. Plan your data file so it fits this expectation to help prevent errors.

See the SDK example for a modeling example.

For additional information on Darwin, contact your local SparkCognition partner for access to the white paper titled: *Darwin - A Neurogenesis Platform*.

Accessing the API

This document describes the SDK and explains how to access the Darwin API and its functionality.
Additional methods to access the Darwin API include:

- through the `https://darwin-api.sparkcognition.com/v1` end point
- optionally, through user created `curl` commands

For additional information on the Darwin API, contact your local SparkCognition partner for access to see the *SparkCognition Darwin API Guide*.

Notes:

- An *API key* is necessary to use the Darwin SDK.
  Contact SparkCognition or your IT manager for an appropriate key.
- All methods return a 2-tuple, for example:

```
(True, <context-dependent-return-object>)
(False, <some-helpful-message>)
```

Darwin SDK interface

Setup Darwin SDK

Perform the following to download and setup the Darwin SDK:

1. Install Python 3.5 or greater. Alternatively, install *Miniconda*, from https://conda.io/miniconda.html.
2. Create a directory to receive the git repository clone.
3. Change (*cd*) into the new directory.
4. Clone the *darwin-sdk* repository:

   ```
   git clone https://github.com/sparkcognition/darwin-sdk
   ```

5. Change into the new root directory of the *darwin-sdk* cloned darwin-sdk project:

   ```
   cd <NewCloneRootDirectory>
   ```

   Note: By default this is the *master* trunk .

6. Download the code:

   ```
   git pull
   ```

7. Setup the SDK:

   ```
   python setup.py install
   ```

   The SDK defaults to using the production URL: https://darwin-api.sparkcognition.com/v1/

Connect to the Darwin interface

1. Obtain an api key.

   To use the Darwin SDK, an API Key is required. A key can be obtained from SparkCognition support or your IT manager. An api_key is a long string, for example:

   ```
   'RsJ74ZS5AvwznbHh0AfVSgrchhS9KxACDy3jefaQnxb9f6QTSDBFmhnGa0cOIWtNAIFRAG9ToOTpi0mnEo3zFA'
   ```

2. Register the api key.

   The purpose of this method is to set a password for an api_key. Each api_key is synonymous with a service. This method must be invoked once for each api_key to establish a password for that key.

   Notes:

   - After successful registration, the service uses `auth_login()` to login as a service.
   - In version 1, the password cannot be changed.

   Example

```
>>> from amb_sdk.sdk import DarwinSdk
>>> s = DarwinSdk()
>>> s.auth_register('asdf', 'iRgwut4kGs0ymULiuKtMd0WFvBYLMWSj16q2uysQeteqH9ssc+\
EETUvcysnPojRpfycLVHa2Il1N1IlrfEk1YMA')
(True,'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MTU1MzM4NjEsImlh\
dCI6MTUxNTUzMDI2MS ... F56xZQiBT-89nrRz1nIXD5LfawHIj_MlUHQqM36vU')
```

3. Login.

   Login as a *service* or *create a user* under the service and login as a user.
   The following explains how to log in as a *service*.

   Notes:

   - Although `Bearer <auth-token>`, returned by `auth_login()`, is used in subsequent calls to validate authenticity, it is not required for each method.
   - The SDK remembers the auth token for the DarwinSdk object. Although an auth token is currently valid for 1 hour, if the DarwinSdk object session life time exceeds 1 hour, the SDK will request another auth token until the session ends.

   Example

```
>>> s.auth_login('asdf', 'iRgwut4kGs0ymULiuKtMd0WFvBYLMWSj16q2uysQeteqH9ssc+EET\
UvcysnPojRpfycLVHa2Il1N1IlrfEk1YMA')
(True,'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MTU1MzQxNzIsImlh\
dCI6MTUxNTUzMD ... UQQfoXqYFKJSoRXXDNPE985-a08cE6_o')
```

   When login (as a service) successfully completes, the SDK can be used to create and model a workflow.

   Note, there are also `auth_register_user()` and `auth_login_user()` methods that allow you to create users and login as a specific user. You can choose to use the SDK as a service or create users underneath the service to partition datasets/models to be owned by specific users. It is more convenient to employ user accounts because the api_key is not necessary for logging in as a user.

   Example

```
>>> s.auth_register_user('atestuser', 'apassword')
(True,
 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiJkNjY0MmJjOC1iMmU5LTQxO\
DctODFlNS00YjI2MD ... 5zMp_1FfxU')

>>> s.auth_login_user('atestuser', 'apassword')
(True,
 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI3NGYzYmUxZS0yOTlmLTRhN\
zMtODU5ZC01NGRmM2F ... u1zGCeCONA')
```

4. Verify the connection.

   The default url in the SDK is `https://darwin-api.sparkcognition.com/v1/`. Use `get_url()` and `set_url()` to verify connection to the right Darwin service.

Darwin SDK methods

URL Get/Set methods

DarwinSdk.**get_url**()

Get Darwin service url.

Parameters: None

Returns:

```
(True, <url-string>)
```

Example

```
In [10]: s.get_url()

Out[10]: (True, 'https://darwin-api.sparkcognition.com/v1/')
```

DarwinSdk.**set_url**(*url, version='v1'*)

Set Darwin service url and version.

Parameters:

- **url** - URL to the Darwin service
- **version** - (optional) defaults to ‘v1’

Returns:

```
(True, \url>) or (False, 'invalid url')
```

Example

```
In [9]: s.set_url('https://darwin-api.sparkcognition.com/v1/')

Out[9]: (True, 'https://darwin-api.sparkcognition.com/v1/')
```

Authentication methods

DarwinSdk.**auth_register**(*password, api_key*)

Register as a service. The purpose of this method is to set a password for an api_key. Each api_key is synonymous with a service. This method is invoked only once for each api_key to establish a password for that key. After registration, the service can use *auth_login*() to login as a service.
Note: In version 1, the password cannot be changed.

Parameters:

- *password* - The service level password
- *api_key* - The api key for the service

Returns:

```
(True, 'Bearer <auth-token>') or (False, <error-message>)
```

`Bearer <auth-token>` is used in subsequent calls to validate authenticity.
The SDK remembers the auth token for the AmkSdk object.
Note: Although an auth token is currently valid for 1 hour, if the AmkSdk object session life time exceeds 1 hour, the SDK will request another auth token until the session ends.

Example

```
In [4]: s.auth_register('asdf', 'RsJ74ZS5AvwznbHh0AfVSgrchhS9KxACDy\
3jefaQnxb9f6QTSDBFmhnGa0cOIWtNAIFRAG9ToOTpi0mnEo3zFA')
Out[4]:
(True,
 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiO...iSdU8xlF4yJk')
```

DarwinSdk.**auth_login**(*password, api_key*)

Login as a service.
Note: A service must have a password set using *auth_register()* to login successfully.

Parameters:

- *password* - The service level password
- *api_key* - The api key for the service

Returns:

```
(True, 'Bearer <auth-token>') or (False, <error-message>)
```

`Bearer \auth-token>` is used in subsequent calls to validate authenticity. The SDK remembers the auth token for the AmkSdk object.
Note: Although an auth token is currently valid for 1 hour, if the AmkSdk object session life time exceeds the 1 hour limit, the SDK will acquire another auth token until the session ends.

Example

```
In [5]: s.auth_login('asdf',
'iRgwut4kGs0ymULiuKtMd0WFvBYLMWSj16q2uysQeteqH9ssc+EETUvcysnPojRpfyc\
LVHa2IlN1IlrfEk1YMA')

Out[5]:

(True,

'Bearer
```

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MTU1MzQxN....')
```

DarwinSdk.**auth_register_user**(username, *password*)

Register a user. This method registers a new user.
Note: You must be logged in as a service to create a user.

Parameters:

- *username* - The new end user's username
- *password* - new end user's password

Returns:

```
(True, 'Bearer <auth-token>') or (False, <error-message>)
```

`Bearer <auth-token>` is used in subsequent calls to validate authenticity. The SDK remembers the auth token for the AmkSdk object.
Note: Although an auth token is currently valid for 1 hour, if the AmkSdk object session life time exceeds the 1 hour limit, the SDK will acquire another auth token until the session ends.

Example

```
In [8]: s.auth_register_user('user1', 'user1-password',
'iRgwut4kGs0ymULiuKtMd0WFvBYLMWSj16q2uysQeteqH9ssc+EETUvcysnPojRpfycLV\
Ha2IlN1IlrfEk1YMA')

Out[8]:

(True,

'Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MTU1MzQyN....')
```

DarwinSdk.**auth_login_user**(*username, password*)

Login as a user.
Note: A user must have a username and password set using **auth_register_user**() to successfully login.

Parameters:

- *username* - The end user's username
- *password* - The service level password

Returns:

```
(True, 'Bearer <auth-token>') or (False, <error-message>)
```

`Bearer <auth-token>` is used in subsequent calls to validate authenticity. The SDK remembers the auth token for the AmkSdk object.
Note: Although an auth token is currently valid for 1 hour, if the AmkSdk object session life time exceeds the 1 hour limit, the SDK will acquire another auth token until the session ends.

Example

```
In [9]: s.auth_login_user('user1', 'user1-password',
'iRgwut4kGs0ymULiuKtMd0WFvBYLMWSj16q2uysQeteqH9ssc+EETUvcysnPojRpfycLV\
Ha2IlN1IlrfEk1YMA')

Out[9]:

(True,

'Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE1MTU1MzQzM....')
```

Job status methods

DarwinSdk.**lookup_job_status***(age=None, status=None)*

Get status information for all jobs belonging to the current user or service.

Parameters:

- *age* - (optional) Filter jobs that are less than *X* units old, for example 3w, 2d, or 1h.

- Optional parameters:
  - *status* - If not specified, returns all jobs.
  - *running*
  - *requested*
  - *complete*
  - *failed*

Returns:

```
(True, <list-of-jobs>) or (False, <error-message>)
```

Example

```
In [6]:  s.lookup_job_status(status='Complete')
Out[6]:
(True,
 [{'artifact_names': None,
   'dataset_names': ['cancer-train'],
   'endtime': '2018-02-01T10:53:50.451598',
   'generations': 0,
   'job_name': 'eeef500d629e4a2185eb8af6e18a83b4',
   'job_type': 'TrainModel',
   'loss': 2.0,
   'model_name': 'cancer-model',
   'percent_complete': 100,
   'starttime': '2018-02-01T10:52:42.280929',
   'status': 'Complete'}])
```

DarwinSdk.**lookup_job_status_name**(*job_name*)

Get job status information for a job by *name*.

Parameters:

- *job_name* - The job name.

Returns:

```
(True, <job-info>) or (False, <error-message>)
```

Example

```
In [19]: s.lookup_job_status_name('eeef500d629e4a2185eb8af6e18a83b4')
Out[19]:
(True,
 {'artifact_names': None,
  'dataset_names': ['cancer-train'],
  'endtime': None,
  'generations': 0,
  'job_type': 'TrainModel',
  'loss': None,
  'model_name': 'cancer-model',
  'percent_complete': 0,
  'starttime': '2018-02-01T10:52:42.280929',
  'status': 'Running'})

In [20]: s.lookup_job_status('Running')
```

Lookup methods

DarwinSdk.**lookup_artifact**(*type=None*)

Get a list of artifacts belonging to the current user or service.

Parameters:

type - (optional) specifies the type of artifact. Values can be 'Model', 'Test', 'Dataset', 'Risk', 'Run'

Returns:

```
(True, <job-info>) or (False, <error-message>)
```

Example:

```
In [30]: s.lookup_artifact('Run')
http://localhost:5000/v1/lookup/artifact
Out[30]:
(True,
 [{'created_at': '2018-02-01T11:09:55.731040',
```

```
   'id': 'b9a9205a-0772-11e8-a003-3b1c8766dad0',
   'mbytes': 0.0,
   'name': '8a63e21030d1483abb0f892963c1728f',
   'type': 'Run'},
  {'created_at': '2018-02-01T11:11:17.560360',
   'id': 'ea6f3f80-0772-11e8-9abe-77bc32e350c5',
   'mbytes': 0.0,
   'name': 'artifact-1',
   'type': 'Run'}]
```

DarwinSdk.**lookup_artifact_name**(*artifact_name*)

Get information for an artifact specified by its name.

Parameters:

- artifact - specifies an artifact by its name

Returns:

```
(True, <job-info>) or (False, <error-message>)
```

Example:

```
In [31]: s.lookup_artifact_name('artifact-1')
Out[31]:
(True,
 {'created_at': '2018-02-01T11:11:17.560360',
  'mbytes': 0.0,
  'name': 'artifact-1',
  'type': 'Run'})
```

DarwinSdk.**lookup_client**()

Get a client's metadata. A client is the current user or service in context.

Parameters: None

Returns:

```
    (True, <client-info>) or (False, <error-message>)
```

Example

```
In [21]: s.lookup_client()
Out[21]:
(True,
 {'job_limit': None,
  'model_limit': None,
  'tier': 0,
  'upload_limit': None,
  'user_limit': None,
  'username': None})
```

DarwinSdk.**lookup_dataset**()

Get the dataset(s) metadata for a user. The user is the current user or service in the current context. This is useful for listing all created datasets.

Parameters: None

Returns:

```
    (True, <list-of-dataset-info>) or (False, <error-message>)
```

Example

```
In [4]: s.lookup_dataset()
Out[4]:
(True,
 [{'categorical': None,
   'imbalanced': None,
   'mbytes': 0.02019977569580078,
   'name': 'unittest-cancer-dataset2',
   'sequential': None,
   'updated_at': '2018-01-31T15:37:28.310994'},
  {'categorical': None,
   'imbalanced': None,
```

```
   'mbytes': 0.02019977569580078,
   'name': 'cancer-train',
   'sequential': None,
   'updated_at': '2018-02-01T10:52:06.076279'}])
```

DarwinSdk.**lookup_dataset_name**(*dataset_name*)

Get a specific dataset's metadata.

Parameters:

- *dataset_name* - The name of the dataset. The name of a dataset is established in the **upload_dataset**() method.

Returns:

```
(True, <dataset-info>) or (False, <error-message>)
```

Example

```
In [36]: s.lookup_dataset_name('cancer-train')
Out[36]:
(True,
 {'categorical': None,
  'imbalanced': None,
  'mbytes': 0.02019977569580078,
  'sequential': None,
  'updated_at': '2018-02-01T10:52:06.076279'})
```

DarwinSdk.**lookup_model**()

Get the model(s) metadata for a user. The user is the current user or service in the current context. This is useful for listing all models.

Parameters: None

Returns:

```
(True, <list-of-model-info>) or (False, <error-message>)
```

Example

```
In [37]: s.lookup_model()
Out[37]:
(True,
 [{'generations': 0,
   'loss': 2.0,
   'name': 'cancer-model',
   'parameters': {'target': 'Diagnosis'},
   'trained_on': ['cancer-train'],
   'updated_at': '2018-02-01T10:53:50.443166'}])
```

DarwinSdk.**lookup_model_name**(*model_name*)

Get a specific model's metadata. The name of a model is established in the *create_model()* method.

Parameters:

- *model_name* - The name of the model

Returns:

```
(True, <model-info>) or (False, <error-message>)
```

Example

```
In [40]: s.lookup_model_name('cancer-model')
Out[40]:
(True,
 {'generations': 0,
  'loss': 2.0,
  'parameters': {'target': 'Diagnosis'},
  'trained_on': ['cancer-train'],
  'updated_at': '2018-02-01T10:53:50.443166'})
```

DarwinSdk.**lookup_tier**()

Get metadata for all tiers. A tier specifies certain usage limits such as *number of models* and *datasets*.

Parameters: None

Returns:

```
(True, <list-of-tier-info>) or (False, <error-message>)
```

Example

```
In [41]: s.lookup_tier()
Out[41]:
(True,
 [{'job_limit': None,
   'model_limit': None,
   'tier': 0,
   'upload_limit': None,
   'user_limit': None},
  {'job_limit': 10000,
   'model_limit': 10000,
   'tier': 1,
   'upload_limit': 10000,
   'user_limit': 1000}])
```

DarwinSdk.**lookup_tier_num**(*tier_num*)

Get a specific tier's metadata. A tier specifies certain usage limits such as the *number of models* or *datasets*.

Parameters:

- *tier_num* - The name of the model

Returns:

```
(True, <tier-info>) or (False, <error-message>)
```

Example

```
In [44]: s.lookup_tier_num(1)
Out[44]:
(True,
 {'job_limit': 10000,
  'model_limit': 10000,
  'tier': 1,
  'upload_limit': 10000,
  'user_limit': 1000})
```

Datasets and artifact methods

DarwinSdk.**upload_dataset**(*dataset*, *dataset_name=None*)

Upload a dataset, model, or a figure.

Parameters:

- *dataset*- Path to dataset
- *dataset_name* - Name to be given to dataset, or defaults to filename

Returns:

```
(True, {dataset_name:<name-given-to-dataset>})  or (False, <error-message>)
```

Example

```
In [5]: s.upload_dataset('sets/cancer_train.csv', 'unittest-cancer-dataset')
Out[5]:
(True,
 {'dataset_name': 'unittest-cancer-dataset'})
```

DarwinSdk.**delete_dataset**(*dataset_name*)

Delete the named dataset.

Parameters:

- *dataset_name* - The name of the dataset to be deleted.

Returns:

```
(True, None)  or (False, <error-message>)
```

Example

```
In [6]: s.delete_dataset('unittest-cancer-dataset')
Out[6]:
(True, None)
```

DarwinSdk.**download_artifact**(*artifact_name*)

Download artifact given its name. The methods that return artifacts are:

- *analyze_data()*
- *analyze_model()*
- *run_model()*
- *create_risk_info()*
  Note: The artifact for *analyze_model()* is a png file.

Parameters:

- *artifact_name* - The name of the artifact to download.

Returns:

```
(True, <path-to-file>) or (True, <python-list>) or (False, <error-message>)
```

Example *run_model()* or prediction artifact

```
In [8]: s.download_artifact('4c4b0833fc1b4edeb689cba86c94a58d')
Out[8]:
Out[8]:
(True,
        ['Diagnosis',
        'BENIGN',
        'BENIGN',
        'BENIGN',
        'BENIGN',
        'MALIGNANT',
        ...
        ]
)
```

Example *analyze_data()* artifact

```
 In [17]: c, r = s.download_artifact('03676cb98d5b48fc89b2d058022781d8')

In [18]: print(r)
  categories      col_name     col_type      max     mean       min  \
0       None  mean_profile    numerical  180.219  95.3869    5.8125
1       None   std_profile    numerical  91.8086  44.3984    24.772
2       None  kurt_profile    numerical  8.06952  1.24083  -1.60483
3       None  skew_profile    numerical  68.1016  5.72565  -1.78189
4       None    mean_dmsnr    numerical  222.421   23.786  0.273411
5       None     std_dmsnr    numerical  110.642   35.272   7.56568
6       None    kurt_dmsnr    numerical  32.1986  6.66098  -3.13927
7       None    skew_dmsnr    numerical  1072.96  79.4015  -1.97698
8        0,1         class  categorical     None     None      None

   num_categories      std
0          3939.0  36.4588
1          4635.0  8.03585
2          4639.0  1.80623
3          4639.0  11.0723
4          4012.0  39.2822
5          4639.0  23.9892
6          4639.0  4.87737
7          4639.0   103.01
8             2.0     None
```

Example *analyze_model()* or prediction artifact

```
  In [5]: s.download_artifact('6e4861de29424cb7ad09e467d1869c17')
  Out[5]:
  (True,
   {'filename': '/var/folders/9r/_wwbjsd17_5b74dww69mtg9h0000gp/T/artifact-\
   0nq69erf.png'})
```

Example *create_risk_info()* artifact

```
In [5]: s.download_artifact('6ec2c1bc6c1244ccb3b03f25ebac5850')
Out[5]:
(True,
 {'filename': '/var/folders/9r/_wwbjsd17_5b74dww69mtg9h0000gp/T/tmp\
 jawjkj4f'})

$ head tmpjawjkj4f
risk
0.0
0.0
1.0
0.0
1.0
```

DarwinSdk.**delete_artifact**(*artifact_name*)

Delete the artifact given its name.

Parameters:

- *artifact_name* - The *name* of the artifact to be deleted.

Returns:

```
(True, None)  or (False, <error-message>)
```

Example

```
In [8]: s.delete_artifact('6c482eac9f894cdb9b0e1e487e41730a')
Out[8]:
(True, None)
```

Modeling and analysis methods

DarwinSdk.**create_model**(*dataset_names, **kwargs*)

Create a model trained on the dataset identified by dataset_name. The name of a model is specified in a parameter in kwargs.
Note: If no name is specified, the model is named with a *uuid-like* name.

Parameters:

- *dataset_names* - A single dataset name as a string or a list of dataset string names to be used for training
- ***kwargs* - variable number of keyword arguments, described in *parameters*.
- *parameters* -
    - *model_name*: The string identifier of the model to be trained. If no name is specified, the model is named with a *uuid-like* name.
    - *job_name*: If no name is specified, the job is named with a *uuid-like* name.
    - *target*: String denoting target prediction column in input data.
    - *Model_name*: The string identifier of the model to be trained.
    - *max_train_time*: Sets the training time for the model in 'HH:MM' format.
      Note: This overrides any values set for *max_generation*.
    - *max_generation*: Expected input/type: *numeric*. Sets the training time for the model in generations. If *max_train_time* is set, this parameter is ignored.
    - *recurrent*: Expected input/type: *true/false*. Enables recurrent connections to be evolved in the model. This option can be useful for timeseries or sequential data.
      Note: This option is automatically enabled if a *datetime* column is detected in the input data. This can result in slower model evolution.
    - *impute*: String alias that indicates how to fill in missing values in input data.

ALIAS	DESCRIPTION	COMPLEXITY
'genetic'	Genetic Fill: Automatically determines the most appropriate fast imputation method using evolutionary methods.	Linear
'ffill'	(Default) Forward Fill: Propagate values forward from one example into the missing cell of the next example. Might be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast
'bfill'	Backward Fill: Propagate values backward from one example into the missing cell of the previous example. Might be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast

ALIAS	DESCRIPTION	COMPLEXITY
'mean'	Mean Fill: Computes the mean value of all non-missing examples in a column to fill in missing examples. The result may or might not be interpretable in terms of the input space for categorical variables.	Linear Fast
'median'	Median Fill: Computes the median value of all non-missing examples in a column to fill in missing examples. While the result is interpretable in terms of the input space for categorical variables, the approach might not be appropriate for non-ordinal data.	Linear Fast
'mode'	Mode Fill: Uses the most common value on a column-by-column basis to fill in missing examples. The result is interpretable for both numerical and categorical variables.	Linear Fast
'spline'	Spline Fill: Interpolation using a spline (piecewise function). Might be useful for timeseries or sequential data.	Linear Fast
'Linear'	Linear Interpolation Fill: Interpolation using a Linear function. Might be useful for timeseries or sequential data.	Linear Fast
'knn'	K-Nearest Neighbors Fill: Fills in missing values by averaging the cell values of the k nearest neighbors in the reduced feature space defined by all non-missing columns.	Polynomial Slow
'rmf'	Robust Matrix Factorization Fill: Computes low-rank matrices L (observations x rank), R (features x rank), and E where X is input data, and $X = LR^T + E$.	Polynomial Slow
'mice'	Multiple Imputation by Chained Equations: First imputes missing values using *Forward Fill*. Then, column-by-column, missing values are reintroduced and regressed upon using the other (non-missing) columns. Continues iteratively.	Polynomial, Iterative Very Slow

- *drop*: Expected input/type: *true/false*. Enables automatic pruning of input columns based on different criteria such as amount of missing data, number of unique values, and standard deviation.
  Note:This automatically drops identifier columns (unique value for each sample) and columns that do not contain sufficient data to aid prediction.

- *max_int_uniques*: Expected input/type: *integer*. Threshold for automatic encoding of categorical variables. If a column contains at least *max_int_uniques* unique values, it is treated as categorical and one hot encoded during preprocessing.

- *max_unique_values*: Expected input/type: *integer*. Threshold for automatic pruning of categorical columns prior to one hot encoding based on the number of unique values.
  Note: If a categorical column contains at least *max_unique_values*, it is dropped during preprocessing prior to one hot encoding.

- *feature_eng*: Enables automatic feature generation. Identifies an appropriate time window and augments input with new features derived in the frequency and time domains.
  Note: Can only be applied to timeseries data. String aliases specify methods for window computation.

ALIAS	DESCRIPTION
None	No feature generation will be applied.
'mi'	Uses mutual information to estimate the window length.
'auc'	(**Default**) Uses autocorrelation to estimate the window length.
'user'	User specified window length: see* window_len*.

- *window_len*: Expected input/type: *integer*. User specified window length for feature generation.
  Note: This parameter is used only in the case that *user* is provided for the *feature_eng* parameter.

- *feature_select*: A number in [0,1] specifying the percentage of numerical features to maintain based on their dependency to the target. Ranks all features using mutual information and drops (1 - feature_select)% of the lowest-ranking features. Default is **1** (keep all features).

- *outlier*: A string alias that indicates the outlier detection to apply during preprocessing.
  Note: Outliers are removed and later filled using imputation.

ALIAS	DESCRIPTION
None	(**Default**) No outlier detection will be applied.
'mad'	Uses Median Absolute Deviation to detect outliers.
'perc'	Uses Percentile-based outlier detection.
'isol'	Uses an Isolation Forest to detect outliers.

- *auto_save_per* (supervised only): Expected input/type: *integer*. Sets the checkpoint frequency. The model creation progress is recorded after every auto_save_per generations.
  Note: If the model is retrained, the model begins from the last recorded checkpoint. The model is automatically saved at the end of evolution.

- *imbalance* (supervised only): Expected input/type: *true/false*. Enables automatic imbalance correction that selectively applies random oversampling, random undersampling, synthetic minority oversampling (SMOTE), or adaptive synthetic sampling (ADASYN) to the input data depending on problem characteristics.

- *clustering*: Expected input/type: *true/false*. Enables clustering for unsupervised problems. If false, detects outliers.

- *n_clusters*: Expected input/type: *integer*. Specifies the number of clusters to be used if clustering is enabled.
  Note: If this value is not provided, the number of clusters will be heuristically determined.

- *anomaly_prior*: Expected input/type: *between [0,1]. *Significance level at which a point is defined as anomalous.
  Note: This parameter is used only for unsupervised problems if clustering is disabled.

Returns:

```
(True, {'job_id': <uuid1>, model_name: <model_name>}) or (False, <error-message>)
```

Example

```
In [10]: s.create_model('cancer-data', target="Diagnosis", model_name="cancer-\
model", max_train_time="00:01", max_generation=100)
Out[10]:
(True,
 {'job_id': 'f5124576a4f34e5c9ab3499770455509',
  'model_name': 'cancer-model'})
```

DarwinSdk.**delete_model**(*model_name*)

Delete a model named by *model_name*.

Parameters:

- *model_name* - Name of the model to be deleted.

Returns:

```
(True, None) or (False, <error-message>)
```

Example

```
In [5]: s.delete_model('unittest-cancer-model')
Out[5]: (True, None)
```

DarwinSdk.**resume_training_model**(*model_name, dataset_names, **kwargs*)

Resume training for a model on the dataset(s) identified by *dataset_names*.

Parameters:

- *dataset_name*- Name of dataset(s) used to train.
- *model_name* - Name of the model to train.
- ***kwargs* - variable number of keyword arguments, described below:.
  - *job_name* - If not specified, a uuid is created as the *job_name*.
  - *max_train_time* - If not specified, the *default* is used.

Returns:

```
(True, { "job_id""<uuid>", "model_name": "<model_name>" })  or (False, <error-message>)
```

Example

```
In [8]: s.resume_training_model('unittest-cancer-model', 'unittest-cancer-\
dataset',  target="Diagnosis", max_train_time="00:01")
Out[8]:
(True, {"job_id": "4e59ffc425e047e1a3b872f1e7396976", "model_name": "unittest-\
cancer-model"})
```

DarwinSdk.**analyze_data**(*dataset_name, **kwargs*)

Analyze the dataset given its *name*.

Parameters:

- *dataset_name* - The name of the dataset to be analyzed.
- ***kwargs* - variable number of keyword arguments, described below:
  - *job_name* - (optional) If not specified, a uuid will be created as the job_name.
  - *artifact_name*: (optional) If not specified, a uuid will be created as the *artifact_name*.
  - *target*: String denoting target prediction column in input data.
  - *impute*: String alias that indicates how to fill in missing values in input data.

ALIAS	DESCRIPTION	COMPLEXITY
‘genetic’	Genetic Fill: Automatically determines the most appropriate fast imputation method using evolutionary methods.	Linear
‘ffill’	**(Default)** Forward Fill: Propagate values forward from one example into the missing cell of the next example. Might be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast
‘bfill’	Backward Fill: Propagate values backward from one example into the missing cell of the previous example. Might be useful for timeseries data, but also applicable for both numerical and categorical data.	Linear Fast

ALIAS	DESCRIPTION	COMPLEXITY
‘mean’	Mean Fill: Computes the mean value of all non-missing examples in a column to fill in missing examples. The result may or might not be interpretable in terms of the input space for categorical variables.	Linear Fast
‘median’	Median Fill: Computes the median value of all non-missing examples in a column to fill in missing examples. While the result is interpretable in terms of the input space for categorical variables, the approach might not be appropriate for non-ordinal data.	Linear Fast
‘mode’	Mode Fill: Uses the most common value on a column-by-column basis to fill in missing examples. The result is interpretable for both numerical and categorical variables.	Linear Fast
‘spline’	Spline Fill: Interpolation using a spline (piecewise function). Might be useful for timeseries or sequential data.	Linear Fast
‘Linear’	Linear Interpolation Fill: Interpolation using a Linear function. Might be useful for timeseries or sequential data.	Linear Fast
‘knn’	K-Nearest Neighbors Fill: Fills in missing values by averaging the cell values of the k nearest neighbors in the reduced feature space defined by all non-missing columns.	Polynomial Slow
‘rmf’	Robust Matrix Factorization Fill: Computes low-rank matrices L (observations x rank), R (features x rank), and E where X is input data, and $X = LR^T + E$.	Polynomial Slow
‘mice’	Multiple Imputation by Chained Equations: First imputes missing values using *Forward Fill*. Then, column-by-column, missing values are reintroduced and regressed upon using the other (non-missing) columns. Continues iteratively.	Polynomial, Iterative Very Slow

- *drop*: Expected input/type: *true/false*. Enables automatic pruning of input columns based on different criteria such as amount of missing data, number of unique values, and standard deviation.
  Note:This automatically drops identifier columns (unique value for each sample) and columns that do not contain sufficient data to aid prediction.

- *max_int_uniques*: Expected input/type: *integer*. Threshold for automatic encoding of categorical variables. If a column contains at least *max_int_uniques* unique values, it is treated as categorical and one hot encoded during preprocessing.

- *max_unique_values*: Expected input/type: *integer*. Threshold for automatic pruning of categorical columns prior to one hot encoding based on the number of unique values.
  Note: If a categorical column contains at least *max_unique_values*, it is dropped during preprocessing prior to one hot encoding.

- *feature_eng*: Enables automatic feature generation. Identifies an appropriate time window and augments input with new features derived in the frequency and time domains.
  Note: Can only be applied to timeseries data. String aliases specify methods for window computation.

ALIAS	DESCRIPTION
None	No feature generation will be applied.
‘mi’	Uses mutual information to estimate the window length.
‘auc’	(**Default**) Uses autocorrelation to estimate the window length.
‘user’	User specified window length: see* window_len*.

- *window_len*: Expected input/type: *integer*. User specified window length for feature generation.
  Note: This parameter is used only in the case that *user* is provided for the *feature_eng* parameter.

- *feature_select*: A number in [0,1] specifying the percentage of numerical features to maintain based on their dependency to the target. Ranks all features using mutual information and drops (1 - feature_select)% of the lowest-ranking features. Default is **1** (keep all features).

- *outlier*: A string alias that indicates the outlier detection to apply during preprocessing.
  Note: Outliers are removed and later filled using imputation.

Returns:

```
(True, {"job_name": <string>, "artifact_name": <string>})  or (False, <error-message>)
```

Example

```
In [10]: s.analyze_data('pulsars', feature_select=1, impute="mean", drop=True, \
target="string", max_int_uniques=15, feature_en="mi", outlier="mad", \
max_unique_values=50)

Out [10]:
(True, {'artifact_id': '2c90c1dc402a4a6da37a139dfc2f7871', \
'job_id': '7871ebb62ad1458da64d800bc73019de'})
```

DarwinSdk.**analyze_model**(*model_name, job_name=None, artifact_name=None*)

Analyze the model given its model name.

Parameters:

- *model_name* - The name of the model to be analyzed.
- *job_name* - (optional) If not specified, a uuid is created as the *job_name*.
- *artifact_name* - (optional) If not specified, a uuid is created as the *artifact_name*.

Returns:

```
(True, {"job_name": <string>, "artifact_name": <string>})  or (False, <error-message>)
```

Example

```
In [5]: s.analyze_model('unittest-cancer-model')
Out [5]:
(True, {'artifact_id': '71a8ae55f2934014b45c13a3975f419c', 'job_id': \
'4e59ffc425e047e1a3b872f1e7396976'})
```

DarwinSdk.**run_model**(*dataset_name, model_name, supervised=True, job_name=None, artifact_name=None*)

Run the model given its name and a dataset to use. Use **upload_dataset**() to upload a data set.

Parameters:

- *dataset_name*The name of a dataset to use for running the model.
- *model_name* - The name of the model to run.
- *supervised* - (optional) If not specified, assumes a *supervised* model.
- *job_name* - (optional) If not specified, a uuid is created as the *job_name*.
- *artifact_name* - (optional) If not specified, a uuid is created as the *artifact_name*.

Returns:

```
(True, {"job_name": <string>, "artifact_name": <string>})  or (False, <error-message>)
```

Example

```
[In [9]: s.run_model('unittest-cancer-testdataset', 'unittest-cancer-model')
Out [9]:
(True, {'artifact_id': '6c482eac9f894cdb9b0e1e487e41730a', 'job_id': \
'1696e03c8165404c8e05685ea68baa3c'})
```

DarwinSdk.**create_risk_info**(*failure_dataset_name, timeseries_dataset_name, **kwargs*)

Create risk information given failure and timeseries data. Use *upload_dataset()* to upload datasets.

Notes concerning risk - Risk is a value used in calculating future events. Risk is calculated using algorithms based on sliding time frames and associated historical data that projects forward in time to predict the likelihood of the event. The outcome of the calculations is that the likelihood of an event occurring within a particular time frame becomes available for use. Note that the risk values are dependent on the quality and extent of the historical data as well as the scope of the timeframe used for evaluation.

Parameters:

- *failure_dataset_name* - The name of a failure dataset.
- *timeseries_dataset_name* - The name of a timeseries dataset.
- ***kwargs* - variable number of keyword arguments, described below:
  - *job_name* - (optional) If not specified, a uuid is created as the *job_name*.
  - *artifact_name* - (optional) If not specified, a uuid is created as the *artifact_name*.
  - *risk_columns*: A list of column names in the index.
  - *shutdown_column*: Name of the column in the risk data that denotes the beginning of the predicted event of interest.
  - *return_column*: Name of the column in the risk data that denotes the end of the predicted event *and* when all data can again be considered "normal".
  - *asset_column*: Name of the asset column in the risk data. This parameter is used when the datasets consist of multiple different assets.
  - *lead_time*: Lead time in seconds. This value is half width of the risk function - that means. the risk index is 0 prior to *2* lead_time* and increases to 1 at a failure time.
  - *Functional_form*: Shape of a risk function, includes:
    - `step`: Step function
    - `linear`: Linear function
    - `sigmoid`: Sigmoid function
    - `exponential`: Exponential function

Returns:

```
(True, {"job_name": <string>, "artifact_name": <string>})  or (False, <error-message>)
```

Example

```
s.create_risk_info('failure-data', 'timeseries-data', return_column=\
"Date Returned to Service", shutdown_column="Shutdown Date", lead_time=1.0, \
functional_form="linear")
```

Convenience methods

DarwinSdk.**delete_all_datasets**()

Deletes user datasets. This method deletes all datasets in the current user or service context.
Note: Use *lookup_dataset()* to view/verify the datasets for deletion.

Parameters: None

Returns:

```
(True, None)  or (False, <error-message>)
```

DarwinSdk.**delete_all_models**()

Delete all models for a user. This method will delete all models in the
current user's or service's context.
Note: Use *lookup_model()* to review and verify that you want to delete all listed models.

Parameters: None

Returns:

```
(True, None)  or (False, <error-message>)
```

DarwinSdk.**wait_for_job***(job_name, time_limit=600)*

Synchronously wait for a job to complete, limited by *time_limit* that defaults to 600 seconds.

Parameters:

- *job_name* - The id for the job
- *time_limit* - (optional) defaults to 600 seconds

Returns::

```
(True, None)  or (False, <error-message>)
```

Reference

SDK modeling example

The following example shows the Darwin SDK performing a modeling process:

```
  ---
 In [1]: from amb_sdk.sdk import DarwinSdk

In [2]: s = DarwinSdk()

In [3]:  s.auth_login_user('username', 'password')
Out[3]:
(True,
 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2ZDZkMTI3Mi0wZDAxLTR\
hYmMtOWYwOC0xYWEwZmYxNDY2NjAiLCJpYXQiOjE1MTYyMTY1MTMsImV4cCI6MTUxNjIyMDExMywi\
dHlwZSI6ImFjY2VzcyIsIm5iZiI6MTUxNjIxNjUxMywiaWRlbnRpdHkiOiJjNTc2NzFjNC1lNTAwL\
TExZTctOWY4ZS1iNzk2ODU2ZTcwMGYiLCJmcmVzaCI6ZmFsc2V9.slh1mYPy_M7DqAok-tV1NT0kU\
4lAgKQoQHk6nYtetg4')

In [12]: s.upload_dataset('sets/cancer_train.csv', 'mydata')
Out[12]: (True, {'dataset_name': 'mydata'})

In [14]: s.create_model('mydata', target="Diagnosis", model_name="my-model")
Out[14]:
(True,
 {'job_name': '1661fb302af149798c34ca9db9e1b0ae', 'model_name': 'my-model'})

In [15]: s.wait_for_job('1661fb302af149798c34ca9db9e1b0ae')
{'percent_complete': 39, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': None, 'loss': 0.4169575273990631, \
'generations': 11, 'status': 'Running', 'starttime': '2018-02-01T14:16:51.4\
64827', 'artifact_names': None}
{'percent_complete': 62, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': None, 'loss': 0.39973780512809753, \
```

```
'generations': 17, 'status': 'Running', 'starttime': '2018-02-01T14:16:51.4\
64827', 'artifact_names': None}
{'percent_complete': 84, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': None, 'loss': 0.39636287093162537, \
'generations': 21, 'status': 'Running', 'starttime': '2018-02-01T14:16:51.4\
64827', 'artifact_names': None}
{'percent_complete': 100, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': '2018-02-01T14:18:02.072976', 'loss': \
0.39636287093162537, 'generations': 23, 'status': 'Complete', 'starttime': \
'2018-02-01T14:16:51.464827', 'artifact_names': None}
Out[15]: (True, 'Job completed')

In [16]: s.upload_dataset('sets/cancer_test.csv', 'mytestdata')
Out[16]: (True, {'dataset_name': 'mytestdata'})

In [19]: s.run_model('mytestdata', 'my-model')
Out[19]:
(True,
 {'artifact_name': '9a6d41532cec47618beee6236b02c129',
  'job_name': '91c7813334ee4c37a733761dce71c0b3'})

In [21]: s.wait_for_job('91c7813334ee4c37a733761dce71c0b3')
{'loss': 0.39636287093162537, 'job_type': 'RunModel', 'artifact_names': \
['9a6d41532cec47618beee6236b02c129'], 'endtime': '2018-02-01T14:22:39.05466', \
'percent_complete': 100, 'generations': 23, 'model_name': 'my-model', 'status'\
: 'Complete', 'starttime': '2018-02-01T14:22:34.219185', 'dataset_names': \
['mytestdata']}
Out[21]: (True, 'Job completed')

In [22]: s.download_artifact('9a6d41532cec47618beee6236b02c129')
(True,      Diagnosis
 0       BENIGN
 1       BENIGN
 2       BENIGN
 3       BENIGN
 4       BENIGN
 5    MALIGNANT
 6    MALIGNANT
 ...
 98   MALIGNANT
 99   MALIGNANT

 [100 rows x 1 columns])
```

SDK analyze data workflow example

The following example shows a Darwin SDK data analysis workflow example:

```


In [1]: from amb_sdk.sdk import DarwinSdk

In [2]: s = DarwinSdk()

In [3]:  s.auth_login_user('username', 'password')
Out[3]:
(True,
 'Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiI2ZDZkMTI3Mi0wZDAxLTRh\
YmMtOWYwOC0xYWEwZmYxNDY2NjAiLCJpYXQiOjE1MTYyMTY1MTMsImV4cCI6MTUxNjIyMDExMywidH\
lwZSI6ImFjY2VzcyIsIm5iZiI6MTUxNjIxNjUxMywiaWRlbnRpdHkiOiJjNTc2NzFjNC1lNTAwLTEx\
ZTctOWY4ZS1iNzk2ODU2ZTcwMGYiLCJmcmVzaCI6ZmFsc2V9.slh1mYPy_M7DqAok-tV1NT0kU4lA\
gKQoQHk6nYtetg4')

In [24]: s.upload_dataset('sets/pulsars.csv', 'pulsars-data')
Out[24]: (True, {'dataset_name': 'pulsars-data'})

In [14]: s.create_model('mydata', target="Diagnosis", model_name="my-model")
Out[14]:
(True,
 {'job_name': '1661fb302af149798c34ca9db9e1b0ae', 'model_name': 'my-model'})

In [15]: s.wait_for_job('1661fb302af149798c34ca9db9e1b0ae')
{'percent_complete': 39, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': None, 'loss': 0.4169575273990631, \
'generations': 11, 'status': 'Running', 'starttime': '2018-02-01T14:16:51.\
464827', 'artifact_names': None}
{'percent_complete': 62, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': None, 'loss': 0.39973780512809753, \
'generations': 17, 'status': 'Running', 'starttime': '2018-02-01T14:16:51.\
464827', 'artifact_names': None}
{'percent_complete': 84, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': None, 'loss': 0.39636287093162537, \
'generations': 21, 'status': 'Running', 'starttime': '2018-02-01T14:16:51.4648\
27', 'artifact_names': None}
{'percent_complete': 100, 'job_type': 'TrainModel', 'model_name': 'my-model', \
'dataset_names': ['mydata'], 'endtime': '2018-02-01T14:18:02.072976', 'loss': \
0.39636287093162537, 'generations': 23, 'status': 'Complete', 'starttime': \
'2018-02-01T14:16:51.464827', 'artifact_names': None}
Out[15]: (True, 'Job completed')

In [16]: s.upload_dataset('sets/cancer_test.csv', 'mytestdata')
Out[16]: (True, {'dataset_name': 'mytestdata'})
```

```
In [17]: s.analyze_data('pulsars-data', feature_select=None, impute="mean", \
drop=True, target=None, max_int_uniques=15, feature_eng="mi", outlier=None, \
max_unique_values=50)
Out[17]:
(True,
 {'artifact_name': '929bd117a07d411ba40d148ddd686d51',
  'job_name': '4df3e87a87224c1993120482b9b00843'})

In [19]: s.wait_for_job('4df3e87a87224c1993120482b9b00843')
{'starttime': '2018-02-01T15:13:05.624744', 'model_name': None, 'dataset_names'\
: ['pulsars-data'], 'artifact_names': ['929bd117a07d411ba40d148ddd686d51'], \
'percent_complete': 100, 'job_type': 'AnalyzeData', 'endtime': '2018-02-01T15:\
13:09.0199', 'generations': None, 'loss': None, 'status': 'Complete'}
Out[19]: (True, 'Job completed')

In [20]: s.download_artifact('929bd117a07d411ba40d148ddd686d51')
Out[20]:
(True,   categories      col_name     col_type      max     mean       min  \
 0       None  mean_profile    numerical  180.219  95.3869    5.8125
 1       None   std_profile    numerical  91.8086  44.3984    24.772
 2       None  kurt_profile    numerical  8.06952  1.24083  -1.60483
 3       None  skew_profile    numerical  68.1016  5.72565  -1.78189
 4       None    mean_dmsnr    numerical  222.421   23.786  0.273411
 5       None     std_dmsnr    numerical  110.642   35.272   7.56568
 6       None    kurt_dmsnr    numerical  32.1986  6.66098  -3.13927
 7       None    skew_dmsnr    numerical  1072.96  79.4015  -1.97698
 8        0,1         class  categorical     None     None      None

    num_categories      std
 0          3939.0  36.4588
 1          4635.0  8.03585
 2          4639.0  1.80623
 3          4639.0  11.0723
 4          4012.0  39.2822
 5          4639.0  23.9892
 6          4639.0  4.87737
 7          4639.0   103.01
 8             2.0     None  )
```

Revision Table

Version	Author	Date	Notes
v 1.0	SparkCognition, SCheng	02.05.2018	First Release