Table of Contents

[How Does it Work? 4](#_Toc9516617)

[Running the Files Using Surround 5](#_Toc9516618)

[**1.** **Project Resources** 5](#_Toc9516619)

[Links to key project files. 5](#_Toc9516620)

[**Revision History (What’s New)** 5](#_Toc9516621)

[**Project Info** 6](#_Toc9516622)

[What is the project? 6](#_Toc9516623)

[Name: 6](#_Toc9516624)

[Purpose 6](#_Toc9516625)

[License 6](#_Toc9516626)

[Who is working? 6](#_Toc9516627)

[Project Leader 6](#_Toc9516628)

[Example Implementation Team 6](#_Toc9516629)

[Sponsor 6](#_Toc9516630)

[How can you find more information? 6](#_Toc9516631)

[Key Contacts 6](#_Toc9516632)

[Project Leaders: 6](#_Toc9516633)

[Mailing List 6](#_Toc9516634)

[Supervisor 7](#_Toc9516635)

[Unit Chair 7](#_Toc9516636)

[Releases Info 7](#_Toc9516637)

[Current Release 7](#_Toc9516638)

[Last Reviewed Release 7](#_Toc9516639)

[Other Releases 7](#_Toc9516640)

[Overview of the Surround 7](#_Toc9516641)

[Why was surround created? 8](#_Toc9516642)

[Why Use Surround. 8](#_Toc9516643)

[Features of Surround. 9](#_Toc9516644)

[Components of Surround 9](#_Toc9516645)

[1. **Surround**: 9](#_Toc9516646)

[2. **Surround Data**: 9](#_Toc9516647)

[3. **Stage**: 10](#_Toc9516648)

[4. **Runner**: 10](#_Toc9516649)

[Libraries Utilized in Surround 10](#_Toc9516650)

[**SkLearn:** 10](#_Toc9516651)

[**NumPy** , **SciPy** , **Matplotlib** , **IPython** , **Sympy** , **Pandas** 11](#_Toc9516652)

[**Sklearn.Preprocessing:** 11](#_Toc9516653)

[**Normalize**: 11](#_Toc9516654)

[**Pandas:** 11](#_Toc9516655)

[**MatplotLib** : 11](#_Toc9516656)

[**Train\_Test\_Split:** 12](#_Toc9516657)

[**Parameters:** 12](#_Toc9516658)

[**Logistic Regression:** 12](#_Toc9516659)

[**Metrics**: 12](#_Toc9516660)

[**Label Encoder:** 12](#_Toc9516661)

[**Time:** 13](#_Toc9516662)

[**Logging:** 13](#_Toc9516663)

Surround AI Version 1.0



*Light-Weight Machine Learning Framework*

*for Data Scientists*

**

*THIS FRAMEWORK IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE*

“At its core, Surround is a lightweight framework for serving machine learning pipelines in Python. It is designed to be flexible, easy to use and to assist data scientists by focusing them on the problem at hand rather than writing glue code.”

# How Does it Work?

First, the input data is processed by three stages which can be manipulated by the users using the config, the first stage involves processing the raw input data to meaningful data.

The surround data is an object which includes useful information which is carry forwarded from one stage to another. Users can also add additional variables for transforming the input data as per their preference. Mainly variables are used for error handling, time required for execution, transformation analysis etc.

Finally, the processed data is implemented in the final stage. In the final stage the surround data is altered to achieve the desired result. Each stage in the process is designed for performing relational activities with the other stages, like the first stage prepares the data which is processed in the last stage of the process.

# Running the Files Using Surround

Once the appropriate version of python is installed in the OS, users can use the surround framework implemented in python code. For that, firstly users must write the code in python and save the code in .py which is an extension for python. For executing the code using surround users need to use the following command:

For instance, the name of the file containing the code is myproject.py. The file needs to be compiled first then executed.

# Project Resources

## Links to key project files.

More information about this project can be found on the GitHub page repository.

Here are the links to the project resources:

<https://github.com/a2i2/surround>

<https://deakin365.sharepoint.com/sites/Surround-AI-Tribe-Squad2>

<https://github.com/Shibi8/Surround_AI_Suqad_2/>

# **Revision History (What’s New)**

The Latest Source release **2.21.0**was released on 24 February 2019

For other previous reviewed released please click on the link to access the history of releases

<https://mirrors.edge.kernel.org/pub/software/scm/git/>

# Project Info

## What is the project?

### Name:

Surround AI

### Purpose

To create a flexible and easy to use machine learning pipelines in python.

Help Data Scientists and software engineers to deploy AI Products in the Surround Framework.

### License

Surround is released under a [BSD-3](https://opensource.org/licenses/BSD-3-Clause) license.

## Who is working?

### Project Leader

SCOTT BARNETT

### Example Implementation Team

Dipesh Bhatt (English Premier League Match Prediction)

Sai Krishna Kesineni (Apple shares Price Prediction)

### Sponsor

Applied Artificial Intelligence Institute

## How can you find more information?

You can find most of the information on surround on the GitHub Repo by accessing the following link:

<https://github.com/a2i2/surround>

## Key Contacts

### Project Leaders:

NIKHIL BHAT

AKSHANT BAJAJ

STEFANUS KURNIAWAN

### Mailing List

[AKHIL PEDDINENI PEDDINENI](mailto:apeddineni@deakin.edu.au);

[BHARATHI BAI BANAVATHU](mailto:bbanavathu@deakin.edu.au);

[DHRUV MILANKUMAR BHATT](mailto:dmbhatt@deakin.edu.au);

[PREMTEJ RAJAYYA;](mailto:prajayy@deakin.edu.au)

[DIPESH BHATTA](mailto:dbhatta@deakin.edu.au);

[HARMILAP SINGH](mailto:harmilapsingh@deakin.edu.au);

[SALMAN KHAN](mailto:khansal@deakin.edu.au);

[DAMINI BEVOORMANDYA JAYARAMU;](mailto:dbevoormandyaja@deakin.edu.au)

[MD IBN SIKBATULLAH](mailto:msikbatullah@deakin.edu.au);

[SAI KRISHNA KESINENI](mailto:skesineni@deakin.edu.au)

[UNIQUE POUDEL](mailto:poudelu@deakin.edu.au)

### Supervisor

[THANH NGUYEN](mailto:duc.nguyen@deakin.edu.au)

### Unit Chair

[ALESSIO BONTI](mailto:a.bonti@deakin.edu.au)

[Mohamed Abdelrazek](mailto:mohamed.abdelrazek@deakin.edu.au)

## Releases Info

### Current Release

The Latest Source release **2.21.0**was released on 24 February 2019

### Last Reviewed Release

For other previous reviewed released please click on the link to access the history of releases

<https://mirrors.edge.kernel.org/pub/software/scm/git/>

### Other Releases

Surround Development team are working progressively to provide a better solution and is under heavy development.

# Overview of the Surround

*Surround* can be a combination of stages or just one depending on the need of the user. The stages can be arranged in any order the user sees fit and these stages process and transform the input data into meaningful output data.

The *surround data* is shareable between and contains information for each stage. What each stage does it read and process the surround data and put back the processed output for the next stage to use that data. This goes one till the final stage which gives us the meaningful output data.

While working with surround data the thing to keep in mind is that a user can create as many variables as he wants but the four core variables that comes with surround data are stage\_metadata, execution\_time, errors and warnings.

*Surround stage* are used for data transformation. It’s the surround stages where the surround data are processed and modified. Each stage can be set to perform a number of actions which helps it process the input data to an extend that the output data of that stage can be used by the next stage. This goes on until the last stage which gives the user desired output.

The number of pipeline implementation is also in the hand of the user which helps him interchange between pipelines.

## Why was surround created?

Surround began as a project at the Applied Artificial Intelligence Institute to address the following problems:

* The same changes were required again and again to refactor code written by data scientists to make it ready for serving e.g. no standard way to run scripts, no standard way to handle configuration and no standard pipeline architecture.
* Existing model serving solutions focus on serving the model rather than serving an end-to-end solution. Our machine learning projects require multiple models and glue code to tie these models together.
* Existing serving approaches do not allow for the evolution of a machine learning pipeline without re-engineering the solution i.e. using a cloud API for the first release before training a custom model much later.
* Code was commonly being commented out to run other branches as experimentation was not a first-class citizen in the code being written.

## Why Use Surround.

* You want a flexible way to serve a pipeline in Python without writing C/C++ code.
* You have multiple models (custom or pre-trained) from different frameworks that need to be combined into a single Surround solution.
* You want to use existing intelligent APIs (AWS Recognition, Google Cloud AI, Cognitive Services) as part of your Surround implementation.
* You have pre or post processing steps that aren't part of your models but need to be deployed as part of your Surround implementation.
* You need to package up your dependencies for running Surround as an offline solution on another machine.

## Features of Surround.

**The features of Surround framework include:**

1. Surround framework is predefined and has structured and modular format of programming that makes understanding easier to those who are new to programming.
2. Each task is performed in Pipeline fashion which makes it easier to understand the program and data flow.
3. Data can be easily manipulated and accessed around the project.
4. As the tasks are divided into stages, error handling is always focused on the stage rather than the entire project.
5. The config.yaml file is accessible via path defined and it is convenient to use throughout the project.
6. Since the project is shared on GitHub, the data or code reusability is high.

With benefits there will always be some drawbacks. Identified are listed below:

1. Plotting a graph or chart is a bit difficult, so as the project progresses, we must focus on that.
2. Necessity of use of operate function in every class is yet to be clarified.

# Components of Surround

The below mentioned components are in this library that can be utilized to build surround solution.

1. Surround

2. Surround Data

3. Stage

4. Runner

* 1. **Surround**:

It is a group of numerous stages or just an initial stage to change raw information into meaningful data. You can set the order of stages directly or by means of a config file. The config file enables you to characterize more than 1 pipeline execution and after that you can switch between them effectively.

* 1. **Surround Data**:

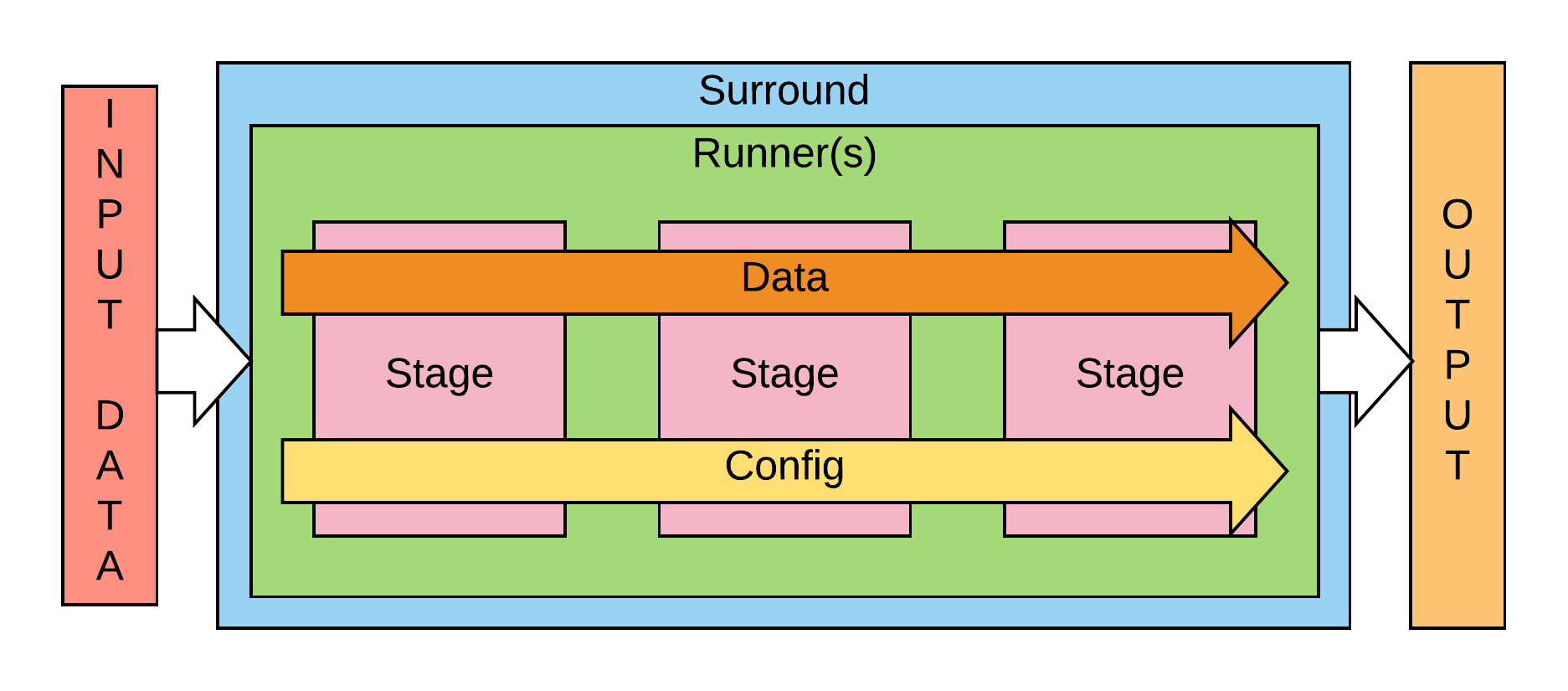
A sharable item between stages that holds vital data for each stage. A phase will read some data from Surround Data, process it, at that point set back new data that will be utilized by different stage(s). When you broaden this class, you can include as many numbers of variables as you require to enable you to change input data into output data. In any case, there are 4 center factors that are being utilized.

* stage\_metadata is information that can be used to identify a stage.
* execution\_time is recorded time to complete a process.
* errors are information to identify failure of a stage.
* warnings are information when transformation is not 100% right.
  1. **Stage**:

A usage of information change. Here is the place Surround Data is altered to accomplish the outcome that you need. Each stage is just meant to execute out a lot of related actions. First stage can be where you get ready information to be prepared and last stage can be the place you populate information to be sent back to the client.

* + - **operate** is a function that you need to override when you extend stage class. It should contain data transformation implementation
  1. **Runner**:

(optional) An interface to connect Surround to/from data.



# Classes and Methods of Surround

Classes are used to create Objects.

Methods are a special kind of function that are defined within a class. Method cannot be called by its name only, we need to invoke the class by a reference of that class in which it is defined, i.e. method is defined within a class and hence they are dependent on that class.

#### 1. class Stage(ABC):

class Stage (ABC):

def dump\_output(self, surround\_data, config):

The method is used to dump the output of each stage.

1. parameter surround\_data is used to store intermediate data from each stage in the pipeline
2. surround\_data must be an instance or child of the SurroundData class
3. parameter config is the Config of the pipeline
4. type config: <class 'surround.config.Config'>

@abstractmethod

def operate(self, surround\_data, config):

A stage in a surround pipeline.

1. parameter surround\_data is used to store intermediate data from each stage in the pipeline
2. surround\_data must be an instance or child of the SurroundData class
3. the parameter config contains the settings for each stage
4. type config: <class 'surround.config.Config'>

def init\_stage(self, config):

This method is used to Initialise stage with some data

1. the parameter config contains the settings for each stage
2. type config: <class 'surround.config.Config'>

#### 2. class Surround (ABC):

class Surround(ABC):

def \_\_init\_\_(self, surround\_stages=None, module=None):

self.surround\_stages = surround\_stages

The collections module has some concrete classes that derive from ABCs; these can be further derived. In addition, the collections.abc submodule has some ABCs that can be used to test whether a class or instance provides a interface, for example, is it hashable or a mapping.

Each abstract base class (ABC) in the collections module provides a common feature (or set of features) with the method functions that are required to implement that feature. In some cases, the features build on each other, and several method functions are required.

Since each of the ABC classes is abstract, they’re missing the implementation of one or more methods. To use these classes, you’ll have to provide the necessary methods.

One very important consequence of using the collections base classes is that it creates standardized names for the various features. This simplifies the assertions that might be required when checking the argument values to a function or method function.

Here, this method is used to carryout initialisation of the class and it can also be used to initialize objects of the class. An instance is created using surround\_stages which is assigned to self.surround\_stages.

if module:

# Module already imported and has a file attribute

mod = sys.modules.get(module)

if mod is not None and hasattr(mod, '\_\_file\_\_'):

package\_path = os.path.dirname(os.path.abspath(mod.\_\_file\_\_))

root\_path = os.path.dirname(package\_path)

else:

raise ValueError("Invalid Python module %s" % module)

self.set\_config(Config(root\_path))

If the module has an attribute, the defined package path is called or accessed, and root\_path is set to the package\_path. Else, a ValueError is thrown with the message "Invalid Python module %s" % module.

#### 3. class Wrapper():

class Wrapper():

def \_\_init\_\_(self, surround, type\_of\_uploaded\_object=None):

self.surround = surround

self.actual\_type\_of\_uploaded\_object = None

A wrapper class wraps an object which it then proxies unhandled calls. Wrapper functions can be used as an interface to adapt to the existing codes, to save from changing current codes back and forth.

#### 4. class AllowedTypes(Enum):

class AllowedTypes(Enum):

JSON = ["application/json"]

FILE = ["file"]

An enumeration is a set of symbolic names (members) bound to unique, constant values. Within an enumeration, the members can be compared by identity, and the enumeration itself can be iterated over.

Here, an Enum class AllowedTypes has been created which specifies the types, with attributes JSON and FILE called enumeration members. These members are constants. If we try and modify any of the members, we will get an error saying Attribute Error.