# InsightSolver API client

*Release 0.1.0*

**Noé Aubin-Cadot**

**Dec 11, 2024**

# CONTENTS:

CHAPTER

ONE

# INTRODUCTION

Welcome to the technical documentation for the **InsightSolver API Client**.

## 1.1 About InsightSolver

InsightSolver is a SaaS (Software as a Service) solution designed for advanced rule mining and data insights. Powered by a centralized rule-mining engine, it enables organizations to uncover hidden patterns and generate actionable insights for data-driven decision-making.

## 1.2 Access Options

InsightSolver can be accessed through the following options:

- The API Client, designed for seamless integration with Python applications and data workflows.
- The **Web App** (not yet available), which will provide an intuitive and interactive interface for exploring rule-mining results, visualization, and analysis.

This documentation specifically covers the API Client. The Web App will have its own documentation once available.

## 1.3 The InsightSolver API client

The InsightSolver API Client offers a Python-based interface for direct interaction with our rule-mining engine. Designed for data engineers, scientists, and developers, this client integrates InsightSolver's functionalities into custom workflows, applications, and automated pipelines. The API client allows users to configure rule-mining parameters, send encrypted data to the server and retrieve results efficiently.

## 1.4 Who Should Use This Documentation?

This documentation is for:

- Developers integrating our rule-mining API.
- Engineers needing setup guides and technical references.
- Users looking for examples of effective API use.

## 1.5 Accessing the API

The API client requires a valid `.json` service key. This key authenticates your identity and ensures secure communication with the InsightSolver API server. If you do not yet have a service key, please contact support@insightsolver.com for assistance.

CHAPTER

TWO

# INSTALLATION

This guide provides instructions to install and set up the InsightSolver API client on your local machine.

## 2.1 Prerequisites

- **Python 3.9 or higher**: Ensure Python is installed on your system. You can check your Python version with:

```
python --version
```

- **pip**: Python's package installer, which is typically included with Python 3 installations.

## 2.2 Installation Steps

You can install the InsightSolver API client in different ways, depending on your setup:

A. **Install directly using pip (100% CLI)**

If you have git installed and don't need a local copy of the repository, run:

```
pip install git+https://github.com/insightsolver/insightsolver.git
```

B. **Clone the repository and install locally (100% CLI)**

If you have git installed and want to also keep a local copy of the repository, run:

```
git clone https://github.com/insightsolver/insightsolver.git
cd insightsolver
pip install .
```

C. **Download via browser and install (50% GUI + 50% CLI)**

If you don't have git installed, follow these steps:

- Open a browser and go to https://github.com/insightsolver/insightsolver.
- Click on the green button <> `Code v` and select `Download ZIP`.
- Extract the ZIP file to a folder on your machine.
- Open a terminal and navigate to the extracted folder using:

```
cd path/to/unzipped-folder
```

- Then install the package with:

```
pip install .
```

## 2.3 Testing the Installation

To verify that the installation was successful, you can run a quick test in Python to ensure all dependencies are correctly installed and functioning.

1. Open a terminal and start a Python interpreter by typing:

   ```
   python
   ```

2. Once inside the Python shell, try importing the `InsightSolver` class with the following command:

   ```
   from insightsolver import InsightSolver
   ```

3. If the installation was successful, there should be no errors, and the Python shell should return to the prompt. If you encounter an *ImportError*, ensure that you have installed the package in the correct environment.

4. Exit the Python shell by typing:

   ```
   quit()
   ```

If you encounter any errors or need assistance, please refer to the *Help section*.

CHAPTER

THREE

# USAGE

The following sections provide examples on how to use the InsightSolver API client.

## 3.1 Quick Start Example

This section provides a quick example of how to use the InsightSolver API client. Before running the example script, please ensure that:

1. You have completed the steps in the *Installation Guide*.
2. You have obtained a valid service key.

The following example demonstrates how to the basic usage of the InsightSolver API client, showing how to initialize the solver and generate insights using the InsightSolver API.

```
# Import some data
import pandas as pd
df = pd.read_csv('kaggle_titanic_train.csv') # Dataset here: https://www.kaggle.com/
↪competitions/titanic/data

# Specify the name of the target variable
target_name = 'Survived' # We are interested in whether the passengers survived or not

# Specify the target goal
target_goal = 1 # We are searching rules that describe survivors

# Choose how features should be interpreted
columns_types = {
                  'Survived' : 'binary',
                  'Pclass'   : 'continuous', # Could be 'multiclass' (i.e. unordered) or
↪'continuous' (i.e. ordered)
                  'Name'     : 'ignore',
                  'Sex'      : 'binary',
                  'Age'      : 'continuous',
                  'SibSp'    : 'continuous', # Could be 'multiclass' (i.e. unordered) or
↪'continuous' (i.e. ordered)
                  'Parch'    : 'continuous', # Could be 'multiclass' (i.e. unordered) or
↪'continuous' (i.e. ordered)
                  'Ticket'   : 'ignore',
                  'Fare'     : 'continuous',
                  'Cabin'    : 'ignore',
                  'Embarked' : 'multiclass',
```

(continues on next page)

(continued from previous page)

```
        }

# Import the class InsightSolver from the module insightsolver
from insightsolver import InsightSolver

# Create an instance of the class InsightSolver
solver = InsightSolver(
   df           = df,           # A dataset
   target_name  = target_name,  # Name of the target variable
   target_goal  = target_goal,  # Target goal
   columns_types = columns_types, # Columns types
)

# Specify the service key
service_key = 'name_of_your_service_key.json'

# Fit the solver
solver.fit(
   service_key = service_key, # Use your API service key here
)

# Print the rule mining results
solver.print(mode='dense')
"""
                        contribution variable              rule     nans
i p_value coverage lift

0 2e-67   19.1%    2.47
                              86.2%      Sex            female
                              13.8%   Pclass            [1, 2]
1 6e-13    6.3%    2.19
                              52.7%      Age       [0.42, 15.0]  exclude
                              47.3%    SibSp            [0, 2]
2 3e-20   12.2%    2.06
                              81.4%   Pclass            [1, 1]
                              11.3%     Fare  [7.925, 512.3292]
                               7.3%      Age        [4.0, 42.0]  exclude
"""
```

In this specific example, the InsightSolver API gives us three rules in which we find more survivors of the Titanic:

- `(i=0)` : **Women in 1st or 2nd class.** This group covers 19.1% of the passengers and has a survival gain of +147% compared to the population of the Titanic.
- `(i=1)` : **Children (which we know the age) with not too many siblings.** This group covers 6.3% of the passengers and has a survival gain of +117% compared to the population of the Titanic.
- `(i=2)` : **Rich 1st class that are not too old (which we know the age).** This group covers 12.2% of the passengers and has a survival gain of +106% compared to the population of the Titanic.

Note that there could be a survivor bias in the two rules `i=1` and `i=2` because we know the age of the survivors more than we know the age of the non-survivors of the Titanic. We could also use `target_goal = 0` to look for passengers that did not survive the Titanic:

```
# Specify the target goal
target_goal = 0 # We are searching rules that describe non-survivors

# Create an instance of the class InsightSolver
solver = InsightSolver(
   df            = df,            # A dataset
   target_name   = target_name,   # Name of the target variable
   target_goal   = target_goal,   # Target goal
   columns_types = columns_types, # Columns types
)

# Fit the solver
solver.fit(
   service_key = service_key,
)

# Print the rule mining results
solver.print(mode='dense')

"""
                        contribution variable            rule     nans
i p_value coverage lift

0 7e-55   42.6%    1.46
                                78.8%      Sex            male
                                10.9%     Fare     [0.0, 26.0]
                                10.3%    Parch          [0, 0]
1 1e-12   12.9%    1.45
                                63.6%     Fare  [7.8875, 15.1]
                                36.4%      Age    [19.0, 26.0]  include
"""
```

In this specific example, the InsightSolver API gives us two rules in which we find more non-survivors of the Titanic:

- `(i=0)` : **Poor males without a family.** This group covers 42.6% of the passengers and has a non-survival gain of +46% compared to the population of the Titanic.
- `(i=1)` : **Poor young adults (include missing ages).** This group covers 12.9% of the passengers and has a non-survival gain of +45% compared to the population of the Titanic.

Note that there could be a survivor bias in the rule `i=1` because we know the age of the non-survivors less than we know the age of the survivors of the Titanic. In conclusion, using the InsightSolver API, we know that *Rose DeWitt Bukater* (young rich female, 1st class, with her family) had a higher chance to survive the Titanic than *Jack Dawson* (3rd class young male without a family). For more technical details about the API, please refer to the detailed documentation.

## 3.2 Advanced usage

This section provides a deeper look at how to use the InsightSolver API client.

Let's revisit the *Titanic* demo. Once the `solver` is fitted, we can do more than simply `print` the results. This becomes particularly important when integrating the InsightSolver API client into a Python pipeline.

### 3.2.1 Conventions

In InsightSolver, the parameter `target_goal` specifies the target modality of the target variable for which rules should capture a large number of 1's. By convention:

- A data point is considered a `1` if it matches the target modality specified by `target_goal`.
- A data point is considered a `0` otherwise.

It is important to note that these `0`'s and `1`'s are conventions used internally by InsightSolver and should not be confused with the actual values or modalities of the target variable in the dataset.

For instance, consider the Titanic dataset used in the *Titanic* example:

- Dataset: `kaggle_titanic_train.csv`.
- Target variable: `target_name='Survived'`.
- Target modalities: `0` (non-survivor), `1` (survivor).
- Target goals: either `target_goal=0` (looking for non-survivors) either `target_goal=1` (looking for survivors).

Here, the modalities `0` and `1` are specific to the Titanic dataset and represent whether a passenger survived or not.

- Total passengers: `891` rows.
- Non-survivors (`Survived=0`): 549 rows.
- Survivors (`Survived=1`): 342 rows.

**Case 1:** Looking for Survivors (`target_goal=1`)

When the goal is to identify survivors:

- `M=891`: Total population.
- `M0=549`: Number of `0`'s, representing non-survivors (`Survived=0`).
- `M1=342`: Number of `1`'s, representing survivors (`Survived=1`).

**Case 2:** Looking for Non-Survivors (`target_goal=0`)

When the goal is to identify non-survivors:

- `M=891`: Total population.
- `M0=342`: Number of `0`'s, representing survivors (`Survived=1`).
- `M1=549`: Number of `1`'s, representing non-survivors (`Survived=0`).

InsightSolver operates under the principle of capturing `1`'s and rejecting `0`'s, regardless of the specific meaning of these values in a given dataset.

### 3.2.2 Attributes of the solver

The `solver` object includes several relevant attributes, which are described exhaustively here. For now, let's take a brief look at the most important ones:

- `M`: The total number of points in the population.
- `M0`: The number of points classified as `0` in the population.
- `M1`: The number of points classified as `1` in the population.
- `rule_mining_results`: A dictionary containing the results of the rule mining process. Below, we'll explore methods to access and parse specific aspects of these results.

- `benchmark_scores`: A dictionary containing the best scores obtained on shuffled data. This is useful to compare the scores of the rules found in the real data against the scores of the rules found in random data.

### 3.2.3 Counting the number of rules

To obtain the number of rules found by the solver, we can use the `ruleset_count` method:

```
solver.ruleset_count() # 3

# 3 rules are found by the solver
```

Each rule in the solver is indexed by an integer, conventionally denoted as `i`.

### 3.2.4 Getting the index of the rules

To retrieve the range of rule indices, we can use the `get_range_i` method:

```
solver.get_range_i() # [0, 1, 2]
```

This shows that the index `i` can take the values `0`, `1` or `2`. Knowing this range is useful when iterating over individual rules in the solver.

### 3.2.5 Exhaustive dictionary of a given rule

Let's take a closer look at the rule at position `i=0`. We can retrieve an exhaustive dictionary of the rule at position `i=0` as follows:

```
    solver.i_to_rule(i=0)

# {
#     "m": 170,
#     "m0": 9,
#     "m1": 161,
#     "coverage": 0.19079685746352412,
#     "m1/M1": 0.47076023391812866,
#     "mu_rule": 0.9470588235294117,
#     "mu_pop": 0.3838383838383838,
#     "sigma_pop": 0.48659245426485753,
#     "lift": 2.4673374613003096,
#     "p_value": 1.925558554763681e-67,
#     "F_score": 0.62890625,
#     "Z_score": 16.767366956025956,
#     "rule_S": {
#         "Sex": "female",
#         "Pclass": [
#             1,
#             2
#         ]
#     },
#     "complexity_S": 2,
#     "F1_pop": 0.5547445255474452,
#     "G_bad_class": 0.17059483726150393,
#     "G_information": 0.24588549145241542,
#     "G_gini": 0.14958927829841417,
```

(continues on next page)

(continued from previous page)

```
#     "p_value_ratio_S": {
#         "Pclass": 5.359920512293736e-08,
#         "Name": 1.0,
#         "Sex": 5.022554571114061e-46,
#         "Age": 1.0,
#         "SibSp": 1.0,
#         "Parch": 1.0,
#         "Ticket": 1.0,
#         "Fare": 1.0,
#         "Cabin": 1.0,
#         "Embarked": 1.0
#     },
#     "F_score_ratio_S": {
#         "Pclass": 1.2812499999999998,
#         "Name": 1.0,
#         "Sex": 0.9302417652027029,
#         "Age": 1.0,
#         "SibSp": 1.0,
#         "Parch": 1.0,
#         "Ticket": 1.0,
#         "Fare": 1.0,
#         "Cabin": 1.0,
#         "Embarked": 1.0
#     },
#     "subrules_S": [
#         {
#             "M": 891,
#             "M0": 549,
#             "M1": 342,
#             "mu_pop": 0.3838383838383838,
#             "sigma_pop": 0.48659245426485753,
#             "F1_pop": 0.5547445255474452,
#             "m": 314,
#             "m0": 81,
#             "m1": 233,
#             "coverage": 0.35241301907968575,
#             "m1/M1": 0.6812865497076024,
#             "mu_rule": 0.7420382165605095,
#             "lift": 1.9332048273550118,
#             "mc": 577,
#             "m0c": 468,
#             "m1c": 109,
#             "p_value": 3.592513266469419e-60,
#             "F_score": 0.7103658536585366,
#             "Z_score": 16.20063097451895,
#             "G_bad_class": 0.17059483726150393,
#             "G_information": 0.21766010666061436,
#             "G_gini": 0.13964795747285225,
#             "complexity": 1,
#             "subrule_S": {
#                 "Sex": "female"
#             },
```

(continues on next page)

(continued from previous page)

```
#             "var_name": "Sex",
#             "var_rule": "female",
#             "p_value_ratio": 5.022554571114061e-46,
#             "shuffling_scores": {
#                 "p_value": {
#                     "cohen_d": 75.86463627446636,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "Z_score": {
#                     "cohen_d": 37.71820414056423,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "F_score": {
#                     "cohen_d": 51.16755087975539,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 }
#             }
#         },
#         {
#             "M": 891,
#             "M0": 549,
#             "M1": 342,
#             "mu_pop": 0.3838383838383838,
#             "sigma_pop": 0.48659245426485753,
#             "F1_pop": 0.5547445255474452,
#             "m": 170,
#             "m0": 9,
#             "m1": 161,
#             "coverage": 0.19079685746352412,
#             "m1/M1": 0.47076023391812866,
#             "mu_rule": 0.9470588235294117,
#             "lift": 2.4673374613003096,
#             "mc": 721,
#             "m0c": 540,
#             "m1c": 181,
#             "p_value": 1.925558554763681e-67,
#             "F_score": 0.62890625,
#             "Z_score": 16.767366956025956,
#             "G_bad_class": 0.17059483726150393,
#             "G_information": 0.24588549145241542,
#             "G_gini": 0.14958927829841417,
#             "complexity": 2,
#             "subrule_S": {
#                 "Sex": "female",
#                 "Pclass": [
#                     1,
#                     2
#                 ]
#             },
```

(continues on next page)

(continued from previous page)

```
#             "var_name": "Pclass",
#             "var_rule": [
#                 1,
#                 2
#             ],
#             "p_value_ratio": 5.359920512293736e-08,
#             "shuffling_scores": {
#                 "p_value": {
#                     "cohen_d": 85.9832032893128,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "Z_score": {
#                     "cohen_d": 39.52974355288319,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 },
#                 "F_score": {
#                     "cohen_d": 23.36359433204899,
#                     "effect_size": "6. huge",
#                     "wy_ratio": 0.0
#                 }
#             }
#         }
#     ],
#     "feature_contributions_S": {
#         "rule_S": {
#             "Sex": "female",
#             "Pclass": "[1, 2]"
#         },
#         "p_value_contribution": {
#             "Sex": 0.8616919700920942,
#             "Pclass": 0.13830802990790583
#         },
#         "F_score_contribution": {
#             "Sex": 1.0,
#             "Pclass": 0.0
#         },
#         "Z_score_contribution": {
#             "Sex": 0.9417181496074654,
#             "Pclass": 0.05828185039253464
#         },
#         "G_bad_class_contribution": {
#             "Sex": 1.0,
#             "Pclass": 0.0
#         },
#         "G_information_contribution": {
#             "Sex": 0.857675593215252,
#             "Pclass": 0.142324406784748
#         },
#         "G_gini_contribution": {
#             "Sex": 0.9099458875412633,
```

(continues on next page)

(continued from previous page)

```
#             "Pclass": 0.0900541124587367
#         }
#     },
#     "shuffling_scores": {
#         "p_value": {
#             "cohen_d": 85.9832032893128,
#             "effect_size": "6. huge",
#             "wy_ratio": 0.0
#         },
#         "Z_score": {
#             "cohen_d": 39.52974355288319,
#             "effect_size": "6. huge",
#             "wy_ratio": 0.0
#         },
#         "F_score": {
#             "cohen_d": 23.36359433204899,
#             "effect_size": "6. huge",
#             "wy_ratio": 0.0
#         }
#     }
# }
```

This dictionary contains detailed information and statistics about the rule at position `i=0`. Here are some of the key entries:

- `"m": 170`: This is the number of points captured by the rule. The rule contains 170 points in total.
- `"m0": 9`: This is the number of `0` captured by the rule. The rule contains 9 non-survivors.
- `"m1": 161`: This is the number of `1` captured by the rule. The rule contains 161 survivors.
- `"coverage": 0.19079685746352412`: This is the coverage of the rule, i.e. the ratio `m/M`. The rule covers 19.1% of the population.
- `"m1/M1": 0.47076023391812866`: This is the sensitivity of the rule, i.e. the capture rate of `1`. The rule captures 47.1% of the survivors.
- `"mu_rule": 0.9470588235294117`: This is the average of the target variable in the rule, i.e. the ratio `m1/m`. Here we have a survival rate of 94.7% in the rule.
- `"mu_pop": 0.3838383838383838`: This the average of the target variable in the population, i.e. the ration `M1/M`. Here we have a survival rate of 38.4% in the population.
- `"sigma_pop": 0.48659245426485753`: This is the standard deviation of the target variable in the population.
- `"lift": 2.4673374613003096`: This is the lift of the rule, i.e. the ratio `mu_rule/mu_pop`.
- `"p_value": 1.925558554763681e-67`: This is the p-value (according to the hypergeometric probability law, not the chi-squared) of the rule.
- `"F_score": 0.62890625`: This is the F1-score of the rule.
- `"Z_score": 16.767366956025956`: This is the Z-score of the rule.
- `"rule_S": {"Sex": "female","Pclass": [1,2]}`: The rule reads *Females in first or second class.*
- `"complexity_S": 2`: The complexity of the rule is 2, i.e. two variables are involded in the rule (`"Sex"` and `"Pclass"`).

- `"F1_pop": 0.5547445255474452`: This is the F1-score of the population.
- `"G_bad_class": 0.17059483726150393`: This is the bad classification gain of the rule.
- `"G_information": 0.24588549145241542`: This is the information gain of the rule.
- `"G_gini": 0.14958927829841417`: This is the Gini gain of the rule.
- `"shuffling_scores"`: This contains the scores that measure how strong is the rule compared to what would be found in shuffled data.

### 3.2.6 DataFrame of subrules

We can retrieve a DataFrame of the subrules for the rule at position `i=0` as follow:

```
    solver.i_to_subrules_dataframe(i=0)

#    p_value_ratio variable    rule  complexity       p_value   F_score  ...  m0c  m1c␣
↪G_bad_class  G_information    G_gini            subrule_S
# 0   5.022555e-46      Sex  female           1  3.592513e-60  0.710366  ...  468  109␣
↪  0.170595       0.217660  0.139648    {'Sex': 'female'}
# 1   5.359921e-08   Pclass  [1, 2]           2  1.925559e-67  0.628906  ...  540  181␣
↪  0.170595       0.245885  0.149589  {'Sex': 'female'...
```

The DataFrame of subrules begins with a rule of complexity 1 (e.g. `{"Sex": "female"}`) and progresses to higher complexities, such as complexity 2 (e.g. `{"Sex": "female","Pclass": [1,2]}`). As we can observe, increasing the complexity from 1 to 2 improves the p-values and the information gain but degrades the F1-score.

The purpose of the subrules DataFrame is to assist in deciding the optimal level of rule complexity based on various metrics.

### 3.2.7 DataFrame of features contributions

We can retrieve a DataFrame showing the contributions of the features for the rule at position `i=0` as follows:

```
    solver.i_to_feature_contributions_S(i=0)

#                p_value  F_score   Z_score  G_bad_class  G_information    G_gini
# feature_name
# Sex           0.861692      1.0  0.941718          1.0       0.857676  0.909946
# Pclass        0.138308      0.0  0.058282          0.0       0.142324  0.090054
```

As we can observe, the variable `"Sex"` provides the largest contribution. The variable `"Pclass"` adds a slight positive contribution to both the p-value and the information gain, as including it in the rule improves these metrics. However, the contribution of `"Pclass"` is zero for the F-score. This indicates that it does not enhance the F-score (in fact, it degrades it, but by convention, contributions are kept nonnegative).

### 3.2.8 Printing modes

Earlier in *Titanic* we saw the `dense` printing mode. There are three printing modes:

- `full`: A full print of the results.
- `light`: A lighter version of the full print.
- `dense`: A very compact version of the print.

### 3.2.9 Column types

The columns of a Pandas DataFrame are associated with a type known as a *dtype*, such as `int64`, `float64`, `object`, and so on. In addition to these, InsightSolver introduces a complementary layer of types called *btype*.

While *dtypes* describe the encoding of the data (e.g., integers or floats), *btypes* define how the data should be interpreted when mining for rules. These *btypes* include:

- `binary`: The variable is treated as a binary categorical variable, and rule mining will focus on finding subsets.
- `multiclass`: The variable is treated as a multiclass categorical variable, and rule mining will aim to find subsets.
- `continuous`: The variable is treated as an ordered variable, and rule mining will focus on identifying meaningful intervals.
- `ignore`: The variable is excluded from rule mining.

The *btypes* of the columns are automatically detected in InsightSolver, so its not mandatory to explicitly specify a *btype* for each variable. However, if the user wishes to specify the *btype* for some or all variables, this can be done using the `columns_types` dictionary (a key is a column name, a value is a *btype*). The `columns_types` dictionary can be passed as a parameter of the `solver`.

CHAPTER

FOUR

# MODULES

## 4.1 insightsolver module

Fichier __init__.py Ici on détermine ce qui est rendu public. On ne va rendre public que la classe InsightSolver car ses méthodes utilisent les autres fonctions.

**Pour appeler la classe on fait :**
from insightsolver import InsightSolver

`class insightsolver.InsightSolver`(*df: DataFrame | None = None*, *target_name: str | int | None = None*, *target_goal: str | Real | uint8 | None = None*, *columns_types: Dict | None = {}*, *columns_descr: Dict | None = {}*, *threshold_M_max: int | None = 10000*, *specified_constraints: Dict | None = {}*, *top_N_rules: int | None = 10*, *verbose: bool = False*)

Bases: `object`

The class `InsightSolver` is meant to :

1. Take input data.
2. Make an insightsolver API calls to the server.
3. Present the results of the rule mining.

### 4.1.1 Attributes

**verbose: bool (default False)**
If we want the initialization to be verbose.

**df: DataFrame**
The DataFrame that contains the data to analyse.

**target_name: str (default None)**
Name of the target variable (by default it's the first column).

**target_goal: (str or int)**
Target goal.

**columns_types: dict**
Types of the columns.

**threshold_M_max: int (default 10000)**
Threshold on the maximum number of observations to consider, above which we under sample the observations to 10000.

**specified_constraints: dict**
Dictionary of the specified constraints on `m_min`, `m_max`, `coverage_min`, `coverage_max`.

**top_N_rules: int (default 10)**
An integer that specifies the maximum number of rules to get from the rule mining.

### 4.1.2 Methods

**validate_class_integrity: None**
Validates the integrity of the class.

**ingest_dict: None**
Ingests a Python dict.

**ingest_json_string: None**
Ingests a JSON string.

**fit: None**
Fits the solver.

**ruleset_count: int**
Counts the number of rules held by the InsightSolver.

**i_to_rule: dict**
Gives the rule i of the InsightSolver.

**i_to_subrules_dataframe: DataFrame**
Returns a DataFrame containing the informations about the subrules of the rule i.

**i_to_feature_contributions_S: DataFrame**
Returns a DataFrame of the feature contributions of the variables in the rule S at position i.

**i_to_print: None**
Prints the content of the rule i in the InsightSolver.

**get_range_i: list**
Gives the range of i in the InsightSolver.

**print: None**
Prints the content of the InsightSolver.

**print_light: None**
Prints the content of the InsightSolver ('light' mode).

**print_dense: None**
Prints the content of the InsightSolver ('dense' mode).

### 4.1.3 Example

Here's a sample code to use the class `InsightSolver`:

```
# Specify the service key
service_key = 'name_of_your_service_key.json'

# Import some data
import pandas as pd
df = pd.read_csv('kaggle_titanic_train.csv')

# Specify the name of the target variable
target_name = 'Survived' # We are interested in whether the passengers survived or␣
↪not
```

(continues on next page)

(continued from previous page)

```
# Specify the target goal
target_goal = 1 # We are searching rules that describe survivors

# Import the class InsightSolver from the module insightsolver
from insightsolver import InsightSolver

# Create an instance of the class InsightSolver
solver = InsightSolver(
        df          = df,          # A dataset
        target_name = target_name, # Name of the target variable
        target_goal = target_goal, # Target goal
)

# Fit the solver
solver.fit(
        service_key = service_key, # Use your API service key here
)

# Print the rule mining results
solver.print()
```

**__init__**(*df: DataFrame | None = None*, *target_name: str | int | None = None*, *target_goal: str | Real | uint8 | None = None*, *columns_types: Dict | None = {}*, *columns_descr: Dict | None = {}*, *threshold_M_max: int | None = 10000*, *specified_constraints: Dict | None = {}*, *top_N_rules: int | None = 10*, *verbose: bool = False*)

The initialization occurs when an `InsightSolver` class instance is created.

### Parameters

**verbose: bool (default False)**
If we want the initialization to be verbose.

**df: DataFrame**
The DataFrame that contains the data to analyse (a target column and various feature columns).

**target_name: str**
Name of the column of the target variable.

**target_goal: str (or other modality of the target variable)**
Target goal.

**columns_types: dict**
Types of the columns.

**columns_descr: dict**
Descriptions of the columns.

**threshold_M_max: int**
Threshold on the maximum number of observations to consider, above which we sample observations.

**specified_constraints: dict**
Dictionary of the specified constraints on m_min, m_max, coverage_min, coverage_max.

**top_N_rules: int (default 10)**
An integer that specifies the maximum number of rules to get from the rule mining.

**target_threshold: float**
Threshold used to convert a continuous target variable to a binary target variable.

**M: int**
Number of points, i.e. number of rows of df.

**M0: int**
Number of points carrying the value 0.

**M1: int**
Number of points carrying the value 1.

**rule_mining_results: dict**
Dictionary that contains the results of the rule mining.

### Returns

**solver: InsightSolver**
An instance of the class InsightSolver.

### Example

Here's a sample code to instantiante the class `InsightSolver`:

```
# Import the class InsightSolver from the module insightsolver
from insightsolver import InsightSolver

# Create an instance of the class InsightSolver
solver = InsightSolver(
        df          = df,          # A dataset
        target_name = target_name, # Name of the target variable
        target_goal = target_goal, # Target goal
)
```

**ingest_dict**(*d: dict*, *verbose: bool = False*) → None

This method aims to ingest a Python dict in the solver.

**ingest_json_string**(*json_string: str*, *verbose: bool = False*) → None

This method aims to ingest a JSON string in the solver.

**fit**(*verbose: bool = False*, *computing_source: str = 'auto'*, *service_key: str | None = None*, *api_source: str = 'auto'*, *do_compress_data: bool = False*) → None

This method aims to fit the solver.

**ruleset_count**() → int

This method returns the number of rules held in the InsightSolver.

**i_to_rule**(*i: int*) → dict

**i_to_subrules_dataframe**(*i: int = 0*) → DataFrame

This method returns a DataFrame which contains the informations about the subrules of the rule i.

**i_to_feature_contributions_S**(*i: int*, *do_rename_cols: bool = True*, *do_ignore_col_rule_S: bool = True*) → DataFrame

This method returns a DataFrame of the feature contributions of the variables in the rule S at position i.

**i_to_print**(*i: int*, *indentation: str = ''*, *do_print_rule_DataFrame: bool = False*, *do_print_subrules_S: bool = True*, *do_show_coverage_diff: bool = False*, *do_print_feature_contributions_S: bool = True*) → None

This method prints the content of the rule i in the InsightSolver.

**get_range_i**(*complexity_max: int | None = None*) → list

This method gives the range of i in the InsightSolver. If the integer complexity_max is specified, return only this number of elements.

**print**(*verbose: bool = False*, *r: int | None = None*, *do_print_dataset_metadata: bool = True*, *do_print_rule_mining_results: bool = True*, *do_print_rule_DataFrame: bool = False*, *do_print_subrules_S: bool = True*, *do_show_coverage_diff: bool = False*, *do_print_feature_contributions_S: bool = True*, *separation_width_between_rules: int | None = 79*, *mode: str = 'full'*) → None

This method prints the content of the InsightSolver.

**print_light**(*print_format: str = 'list'*) → None

This method does a 'light' print of the InsightSolver.

Two formats: - 'list': shows the rules via a loop of prints. - 'compact': shows the rules in a single DataFrame.

**print_dense**() → None

This method is aimed at printing a 'dense' version of the InsightSolver object.

## 4.2 api_utilities module

- *Project Name*: InsightSolver
- *Module Name*: api_utilities
- *Author*: Noé Aubin-Cadot
- *Organization*: InsightSolver
- *Email*: noe.aubin-cadot@insightsolver.com
- *Last Updated*: 2024-11-18
- *First Created*: 2024-09-16

### 4.2.1 Description

This module provides essential utility functions to secure and streamline client-server communication within the API. It includes functions for data compression, encryption, decryption, and transformations of data structures, all designed to facilitate efficient and protected message exchange between the client and server.

While all communications are secured via HTTPS, this module goes a step further by adding an additional layer of encryption, using RSA-4096 and ECDSA-SECP521R1 for secure key exchange and AES-256 for data encryption. These functions are particularly useful for scenarios requiring enhanced data privacy and integrity.

### 4.2.2 Functions provided

- `convert_bytes_to_base64_string`: Convert bytes to a base64 string.
- `convert_base64_string_to_bytes`: Convert a base64 string to bytes.
- `compress_string`: Compress a string using gzip.
- `decompress_string`: Decompress a gzip-compressed string.

- `compress_and_encrypt_string`: Compress and encrypt a string for secure transmission.
- `decrypt_and_decompress_string`: Decrypt an encrypted string.
- `transform_dict`: Convert a dictionary for easier client-server communication.
- `untransform_dict`: Reverse the dictionary transformation to restore the original data format.
- `request_cloud_public_keys`: Request the server for public keys.
- `request_cloud_computation`: Request the server for computation.
- `generate_keys`: Generate RSA and ECDSA private and public keys.
- `search_best_ruleset_from_API_dict`: Make the API call.

### 4.2.3 License

Exclusive Use License - see LICENSE for details.

---

`api_utilities.`**`convert_bytes_to_base64_string`**(*data: bytes*) → str

Convert a bytes object to a base64-encoded string.

### 4.2.4 Parameters

**data**
: [bytes] The byte data to encode.

### 4.2.5 Returns

**str**
: The base64-encoded string.

`api_utilities.`**`convert_base64_string_to_bytes`**(*string: str*) → bytes

Convert a base64-encoded string to a bytes object.

### 4.2.6 Parameters

**string**
: [str] The base64-encoded string.

### 4.2.7 Returns

**bytes**
: The decoded byte data.

`api_utilities.`**`compress_string`**(*original_string: str*) → str

Compress a string using gzip and then encode it to base64.

### 4.2.8 Parameters

**original_string**
: [str] The original string to be compressed.

#### 4.2.9 Returns

**str**
The compressed string.

#### 4.2.10 Example

```
original_string = "This is a test string"
compressed_string = compress_string(original_string)
print(compressed_string)  # Example output: 'H4sIAA01/2YC/
↪wvJyCxWAKJEhZLU4hKF4pKizLx0AG3zTmsVAAAA'
```

api_utilities.**decompress_string**(*compressed_string: str*) → str

Decompress a base64-encoded string that was previously compressed using gzip.

This function takes a base64-encoded string, decodes it, and then decompresses the resulting data using gzip to return the original string.

#### 4.2.11 Parameters

**compressed_string**
[str] The base64-encoded string that contains the compressed data.

#### 4.2.12 Returns

**str**
The original uncompressed string.

#### 4.2.13 Example

```
compressed_string = 'H4sIAA01/2YC/wvJyCxWAKJEhZLU4hKF4pKizLx0AG3zTmsVAAAA'
original_string = decompress_string(compressed_string)
print(original_string) # 'This is a test string'
```

api_utilities.**compress_and_encrypt_string**(*original_string: str*, *symmetric_key: bytes*) → tuple[str, str]

Compress and encrypt a string using AES-256-GCM.

This function compresses the given string using gzip and then encrypts it using AES-256 in GCM mode. A nonce is used in the encryption process for AES-GCM, and the result is base64-encoded for easy transfer over networks.

Security: - AES-256 encryption - GCM (Galois/Counter Mode) with authentication

#### 4.2.14 Parameters

**original_string**
[str] The original string to be compressed and encrypted.

**symmetric_key**
[bytes] The 32-byte symmetric key used for encryption.

#### 4.2.15 Returns

**tuple[str, str]**
A tuple containing the base64-encoded encrypted compressed string and the base64-encoded nonce used.

#### 4.2.16 Example

```
transformed_string, nonce_string = compress_and_encrypt_string(
        original_string = "Secret data",
        symmetric_key   = token_bytes(32),
)
print(transformed_string, nonce_string) # 'Base64_encoded_result', nonce_string
```

api_utilities.**decrypt_and_decompress_string**(*transformed_string: str*, *symmetric_key: bytes*, *nonce: bytes*) → str

Decrypt and decompress a string using AES-256-GCM.

This function takes a base64-encoded encrypted string, decrypts it using AES-256 in GCM mode with the provided symmetric key and nonce, and then decompresses the result using gzip.

Security: - AES-256 encryption - GCM (Galois/Counter Mode) with authentication

#### 4.2.17 Parameters

**transformed_string**
[str] The base64-encoded string that contains the encrypted and compressed data.

**symmetric_key**
[bytes] The 32-byte symmetric key used for decryption.

**nonce**
[bytes] The nonce used for AES-GCM during encryption.

#### 4.2.18 Returns

**str**
The original uncompressed and decrypted string.

#### 4.2.19 Raises

**Exception**
If the decryption fails.

#### 4.2.20 Example

```
original_string = decrypt_and_decompress_string(
        transformed_string = encrypted_compressed_string,
        symmetric_key      = token_bytes(32),
        nonce              = nonce
)
print(original_string) # 'Secret data'
```

api_utilities.**transform_dict**(*d_original: dict*, *do_compress_data: bool = False*, *symmetric_key: bytes | None = None*, *json_format: str = 'json'*) → dict

Transform the contents of a dictionary by optionally compressing and encrypting its data.

This function takes a dictionary and converts it to a string. Depending on the options provided, it can compress the data using gzip, encrypt it using AES-256, or both. The resulting string is returned in a transformed dictionary format for easier transmission or storage.

### 4.2.21 Parameters

**d_original**
: [dict] The original dictionary that needs to be transformed.

**do_compress_data**
: [bool, optional] Whether or not to compress the dictionary data (default is False).

**symmetric_key**
: [bytes, optional] A symmetric key. Typically generated using from secrets import token_bytes;symmetric_key = token_bytes(32). If provided, the data will be encrypted (default is None).

**json_format**
: [str, optional] The format to convert the dictionary to a string. Can be 'json' or 'jsonpickle' (default is 'json').

### 4.2.22 Returns

**dict**
: A dictionary containing the transformed string, the transformations applied, and the json format.

### 4.2.23 Example

```
d_original = {'A':1, 'B':2, 'C':3}
from secrets import token_bytes
symmetric_key = token_bytes(32) # b'\x1a\xef&\x0bR\xe1\x95\xfa\x90\x10r\x93\x1a\xaeN\
↪xc2\xba\x80\xf1\x1a\x0fG\xf4(\x0e#\xd4\xaf`\x81q\xf4'
d_transformed = transform_dict(
        d_original       = d_original,
        do_compress_data = True,
        symmetric_key    = symmetric_key,
        json_format      = 'json',
)
print(d_transformed)
# {
#       'transformations': 'encrypted_gzip_base64',
#       'json_format': 'json',
#       'transformed_string':
↪'q30qPkK19Z3sENnfk77t4CnpzWKV+gdHLLSpNNgU3DjdmEbLcZWj+AjZyFmUquuUmh6obZmTh8k=',
#       'nonce_string': '7PpTvoc0Ksx8whRy',
# }
```

api_utilities.**untransform_dict**(*d_transformed: dict*, *symmetric_key: bytes | None = None*, *verbose: bool = False*) → dict

Decompress and decrypt the contents of a transformed dictionary.

This function takes a dictionary that has been transformed (e.g., compressed, encrypted), and restores its original contents by reversing the transformations. Depending on the transformation type, it may decrypt and/or decompress the data.

#### 4.2.24 Parameters

**d_transformed**
: [dict] The transformed dictionary containing the compressed/encrypted string, the transformations applied, and the json format used.

**symmetric_key**
: [bytes, optional] A symmetric key. Typically generated using `from secrets import token_bytes; symmetric_key = token_bytes(32)`. If provided, the data will be decrypted using this key (default is None).

**verbose**
: [bool, optional] If True, additional debug information will be printed (default is False).

#### 4.2.25 Returns

**dict**
: The original dictionary with its content restored.

#### 4.2.26 Raises

**Exception**
: If an invalid transformation type or JSON format is provided.

#### 4.2.27 Example

```
d_transformed = {
        'transformations'    : 'encrypted_gzip_base64',
        'json_format'        : 'json',
        'transformed_string' :
↪'q30qPkK19Z3sENnfk77t4CnpzWKV+gdHLLSpNNgU3DjdmEbLcZWj+AjZyFmUquuUmh6obZmTh8k='
        'nonce_string'       : '7PpTvoc0Ksx8whRy',
}
d_untransformed = untransform_dict(
        d_transformed = d_transformed,
        symmetric_key = symmetric_key, # b'\x1a\xef&\x0bR\xe1\x95\xfa\x90\x10r\x93\
↪x1a\xaeN\xc2\xba\x80\xf1\x1a\x0fG\xf4(\x0e#\xd4\xaf^\x81q\xf4'
)
print(d_untransformed) # {'A': 1, 'B': 2, 'C': 3}
```

api_utilities.**request_cloud_public_keys**(*computing_source: str*, *d_client_public_keys: dict*, *input_file_service_key: str | None = None*, *timeout: int = 60*) → dict

Send the client's public keys to the server and receive the server's public keys in response.

This function establishes a secure connection to the specified server (`computing_source`) and sends the client's public keys (`d_client_public_keys`). The server responds with its own set of public keys, which are returned in a dictionary format.

#### 4.2.28 Parameters

**computing_source**
: [str] Where the server is.

**d_client_public_keys**
: [dict] A dictionary containing the client's public keys to be sent to the server. The dictionary format is:

- `alice_rsa_public_key_pem_base64`: Client's RSA public key, encoded in base64.
- `alice_ecdsa_public_key_pem_base64`: Client's ECDSA public key, encoded in base64.

**input_file_service_key**
[optional] The client's service key, needed if the server is remote. Default is *None*.

**timeout**
[int, optional] The timeout duration for the request, in seconds. Default is 60 seconds, as this operation is typically fast and does not involve computation.

### 4.2.29 Returns

**dict**
A dictionary containing the server's public keys and a unique session identifier. The dictionary format is as follows:

- `session_id`: A unique identifier for the session.
- `bob_rsa_public_key_pem_base64`: Server's RSA public key, encoded in base64.
- `bob_ecdsa_public_key_pem_base64`: Server's ECDSA public key, encoded in base64.

### 4.2.30 Example

```
# Client's public keys
d_client_public_keys = {
        'alice_rsa_public_key_pem_base64': '<base64-encoded RSA public key>',
        'alice_ecdsa_public_key_pem_base64': '<base64-encoded ECDSA public key>',
}

# Request server public keys
d_server_public_keys = request_cloud_public_keys(
        computing_source='https://server-address.com',
        d_client_public_keys=d_client_public_keys,
        input_file_service_key='client_service_key'
)

# Access the session ID and server's public keys
session_id = d_server_public_keys['session_id']
bob_rsa_public_key = d_server_public_keys['bob_rsa_public_key_pem_base64']
bob_ecdsa_public_key = d_server_public_keys['bob_ecdsa_public_key_pem_base64']
```

### 4.2.31 Raises

**Exception**
If the request fails or the server does not return the expected keys.

api_utilities.**request_cloud_computation**(*computing_source: str*, *d_out_transformed: dict*, *input_file_service_key: str | None = None*, *timeout: int = 600*) → Response

Send the transformed dict to the server for it to compute the rule mining.

#### 4.2.32 Parameters

**computing_source**
[str] The computing source.

**d_out_transformed**
[dict] The transformed dict to send to the server.

**input_file_service_key**
[str, optional] The client's service key, needed if the server is remote. Default is *None*.

**timeout**
[int, optional] Timeout for the request, in seconds. Default is 600 seconds, as computation may take longer.

#### 4.2.33 Returns

**requests.Response**
The response from the server after attempting the computation request.

#### 4.2.34 Raises

**requests.exceptions.RequestException**
If the request to the server fails due to a network issue or server error.

api_utilities.**generate_keys**()

This function generates RSA and ECDSA private and public keys. The generated keys:

- `rsa_private_key`
- `ecdsa_private_key`
- `rsa_public_key_pem_bytes`
- `ecdsa_public_key_pem_bytes`

#### 4.2.35 Returns

**tuple**
A tuple containing four elements:

- rsa_private_key: The generated RSA private key.
- ecdsa_private_key: The generated ECDSA private key.
- rsa_public_key_pem_bytes: The RSA public key serialized in PEM format.
- ecdsa_public_key_pem_bytes: The ECDSA public key serialized in PEM format.

api_utilities.**search_best_ruleset_from_API_dict**(*d_out_original: dict*, *input_file_service_key: str | None = None*, *computing_source: str = 'remote_cloud_function'*, *do_compress_data: bool = True*, *do_compute_memory_usage: bool = True*, *verbose: bool = False*) → dict

Search for the best ruleset where the computation is done from the server.

#### 4.2.36 Parameters

**d_out_original**
: [dict] The original dict, pre-transformation, that contains the necessary data for the server to do rule mining.

**input_file_service_key**
: [str, optional] The service key of the client.

**computing_source**
: [str, optional] The computing source.

**do_compress_data**
: [bool, optional] If we want to compress the data to reduce transmission size.

**do_compute_memory_usage**
: [bool, optional] If we want to compute the memory usage.

**verbose**
: [bool, optional] Verbosity.

#### 4.2.37 Returns

**dict**
: The dict that contain the output of the rule mining from the server.

CHAPTER

FIVE

# HELP

If you need assistance with the InsightSolver API client, please follow the resources below in the suggested order:

## 5.1 1. Technical Documentation

The technical documentation provides detailed guidance on installation, usage, and troubleshooting. Start here if you are encountering issues.

- **Installation Guide**: Refer to the *Installation Guide* for step-by-step instructions.
- **Quickstart Usage Example**: Check out the *Quickstart Example* for a simple and practical example to get started quickly. This example demonstrates how to use the InsightSolver API client and may inspire you to adapt it to your specific needs.
- **API Reference**: Explore the *API Reference* for detailed information on available methods and parameters.

## 5.2 2. Frequently Asked Questions (FAQ)

The FAQ section addresses common questions and issues.

- **How do I install the API client?**

  Please see the *Installation Guide*.

- **What should I do if I encounter a connection error?**

  Ensure your service key is valid, and check that your network permits outgoing connections.

## 5.3 3. GitHub Issues

For reporting bugs or exploring existing issues, visit our GitHub Issues page.

## 5.4 4. Contact Support

If the above resources do not resolve your issue, you can contact us directly at:

- **Email**: support@insightsolver.com

CHAPTER

SIX

# LICENSE

```
Exclusive Use License:

This script is provided under an exclusive use license.
The holder of this license is authorized to use the script for internal
purposes only. Any modification, distribution, or sharing of the script or
parts thereof is strictly prohibited without prior written consent from the author.

This script is provided "as is", without warranty of any kind, express
or implied, including but not limited to the warranties of merchantability,
fitness for a particular purpose, or non-infringement.
```

# PYTHON MODULE INDEX

# INDEX

## Symbols



## A



## C



## D



## F



## G



## I



## M



## P



## R



## S



## T



## U