API Documentation

API Documentation

March 29, 2012

Contents

1 Module attrgenfunct

1.1 Functions

generate_phone_number_australia()

Randomly generate an Australian telephone number made of a two-digit area code and an eight-digit number made of two blocks of four digits (with a space between). For example: '02 1234 5678'

For details see: http://en.wikipedia.org/wiki/Telephone_numbers_in_Australia#Personal_numbers_.2805.29

generate_credit_card_number()

Randomly generate a credit card made of four four-digit numbers (with a space between each number group). For example: '1234 5678 9012 3456'

For details see: http://en.wikipedia.org/wiki/Bank_card_number

generate_uniform_value(*min_val*, *max_val*, *val_type*)

Randomly generate a numerical value according to a uniform distribution between the minimum and maximum values given.

The value type can be set as 'int', so a string formatted as an integer value is returned; or as 'float1' to 'float9', in which case a string formatted as floating-point value with the specified number of digits behind the comma is returned.

Note that for certain situations and string formats a value outside the set range might be returned. For example, if min_val=100.25 and val_type='float1' the rounding can result in a string value '100.2' to be returned.

Suitable minimum and maximum values need to be selected to prevent such a situation.

generate_uniform_age(*min_val*, *max_val*)

```
Randomly generate an age value (returned as integer) according to a
uniform distribution between the minimum and maximum values given.

This function is simple a shorthand for:

  generate_uniform_value(min_val, max_val, 'int')
```

generate_normal_value(*mu*, *sigma*, *min_val*, *max_val*, *val_type*)

Randomly generate a numerical value according to a normal distribution with the mean (mu) and standard deviation (sigma) given.

A minimum and maximum allowed value can given as additional parameters, if set to None then no minimum and/or maximum limit is set.

The value type can be set as 'int', so a string formatted as an integer value is returned; or as 'float1' to 'float9', in which case a string formatted as floating-point value with the specified number of digits behind the comma is returned.

generate_normal_age(*mu*, *sigma*, *min_val*, *max_val*)

```
Randomly generate an age value (returned as integer) according to a
normal distribution following the mean and standard deviation values
given, and limited to age values between (including) the minimum and
maximum values given.

This function is simple a shorthand for:

  generate_normal_value(mu, sigma, min_val, max_val, 'int')
```

1.2 Variables

Name	Description
num_test	**Value:** `20`
__package__	**Value:** `None`

2 Module basefunctions

2.1 Functions

check_is_not_none(*variable*, *value*)

Check if the value given is not None.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_string(*variable*, *value*)

Check if the value given is of type string.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_unicode_string(*variable*, *value*)

Check if the value given is of type unicode string.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_string_or_unicode_string(*variable*, *value*)

Check if the value given is of type string or unicode string.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_non_empty_string(*variable*, *value*)

Check if the value given is of type string and is not an empty string.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_number(*variable*, *value*)

Check if the value given is a number, i.e. of type integer or float.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_positive(*variable*, *value*)

Check if the value given is a positive number, i.e. of type integer or float, and larger than zero.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_not_negative(*variable*, *value*)

Check if the value given is a non-negative number, i.e. of type integer or float, and larger than or equal to zero.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_normalised(*variable*, *value*)

Check if the value given is a number, i.e. of type integer or float, and between (including) 0.0 and 1.0.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_percentage(*variable*, *value*)

Check if the value given is a number, i.e. of type integer or float, and between (including) 0 and 100.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_integer(*variable*, *value*)

Check if the value given is an integer number.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_float(*variable*, *value*)

Check if the value given is a floating-point number.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_dictionary(*variable*, *value*)

Check if the value given is of type dictionary.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_list(*variable*, *value*)

Check if the value given is of type dictionary.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_set(*variable*, *value*)

Check if the value given is of type set.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_tuple(*variable*, *value*)

Check if the value given is of type tuple.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_flag(*variable*, *value*)

Check if the value given is either True or False.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_is_function_or_method(*variable*, *value*)

Check if the value given is a function or method.

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

check_unicode_encoding_exists(*unicode_encoding_str*)

A function which checks if the given Unicode encoding string is known to the Python codec registry.

If the string is unknown this functions ends with an exception.

char_set_ascii(*s*)

Determine if the input string contains digits, letters, or both, as well as whitespaces or not.

Returns a string containing the set of corresponding characters.

check_is_valid_format_str(*variable*, *value*)

Check if the value given is a valid formatting string for numbers. Possible formatting values are:

int, float1, float2, float3, float4, float5, float6, float7, float8, or float9

The argument 'variable' needs to be set to the name (as a string) of the value which is checked.

float_to_str(*f*, *format_str*)

Convert the given floating-point (or integer) number into a string according to the format string given.

The format string can be one of 'int' (return a string that corresponds to an integer value), or 'float1', 'float2', ..., 'float9' which returns a string of the number with the specified number of digits behind the comma.

str2comma_separated_list(*s*)

A function which splits the values in a list at commas, and checks all values if they are quoted (double or single) at both ends or not. Quotes are removed.

Note that this function will split values that are quoted but contain one or more commas into several values.

read_csv_file(*file_name*, *encoding*, *header_line*)

```
Read a comma separated values (CSV) file from disk using the given Unicode
encoding.

Arguments:
file_name    Name of the file to read.

encoding      The name of a Unicode encoding to be used when reading the
              file.
              If set to None then the standard 'ascii' encoding will be
              used.

header_line  A flag, set to True or False, that has to be set according
              to if the frequency file starts with a header line or not.

This function returns two items:
- If given, a list that contains the values in the header line of the
  file. If no header line was given, this item will be set to None.

- A list containing the records in the CSV file, each as a list.

Notes:
- Lines starting with # are assumed to contain comments and will be
  skipped. Lines that are empty will also be skipped.
- The CSV files must not contain commas in the values, while values
  in quotes (double or single) can be handled.
```

write_csv_file(*file_name*, *encoding*, *header_list*, *file_data*)

```
Write a comma separated values (CSV) file to disk using the given Unicode
encoding.

Arguments:
file_name    Name of the file to write.

encoding     The name of a Unicode encoding to be used when reading the
             file.
             If set to None then the standard 'ascii' encoding will be
             used.

header_list  A list containing the attribute (field) names to be written
             at the beginning of the file.
             If no header line is to be written then this argument needs
             to be set to None.

file_data    A list containing the records to be written into the CSV
             file. Each record must be a list of values, and these values
             will be concatenated with commas and written into the file.
             It is assumed the values given do not contain comas.
```

2.2 Variables

Name	Description
__package__	**Value:** `None`

3 Module contdepfunct

3.1 Functions

blood_pressure_depending_on_age(*age*)

Randomly generate a blood pressure value depending upon the given age value.

It is assumed that for a given age value the blood pressure is normally distributed with an average blood pressure of 75 at birth (age 0) and of 90 at age 100, and standard deviation in blood pressure of 4.

salary_depending_on_age(*age*)

Randomly generate a salary value depending upon the given age value.

It is assumed that for a given age value the salary is uniformly distributed with an average salary of between 20,000 at age 18 (salary will be set to 0 if an age is below 18) and 80,000 at age 60.

The minimum salary will stay at 10,000 while the maximum salary will increase from 30,000 at age 18 to 150,000 at age 60.

3.2 Variables

Name	Description
__package__	**Value:** `None`

4 Module corruptor

Module containing several classes to corrupt synthetic data according to user specification.

4.1 Functions

position_mod_uniform(*in_str*)

Select any position in the given input string with uniform likelihood.

Return 0 is the string is empty.

position_mod_normal(*in_str*)

Select any position in the given input string with normally distributed likelihood where the average of the normal distribution is set to one character behind the middle of the string, and the standard deviation is set to 1/4 of the string length.

This is based on studies on the distribution of errors in real text which showed that errors such as typographical mistakes are more likely to appear towards the middle and end of a string but not at the beginning.

Return 0 is the string is empty.

4.2 Variables

Name	Description
__package__	**Value:** `None`

4.3 Class CorruptValue

Known Subclasses: corruptor.CorruptCategoricalValue, corruptor.CorruptMissingValue, corruptor.CorruptValueEdit, corruptor.CorruptValueKeyboard, corruptor.CorruptValueOCR, corruptor.CorruptValuePhonetic

```
Base class for the definition of corruptor that is applied on a single
attribute (field) in the data set.

This class and all of its derived classes provide methods that allow the
definition of how values in a single attribute are corrupted (modified)
and the parameters necessary for the corruption process.

The following variables need to be set when a CorruptValue instance is
initialised (with further parameters listed in the derived classes):

position_function  A function that (somehow) determines the location
                   within a string value of where a modification
                   (corruption) is to be applied. The input of this
                   function is assumed to be a string and its return value
```

```
                  an integer number in the range of the length of the
                  given input string.
```

4.3.1 Methods

__init__(*self*, *base_kwargs*)

Constructor, set general attributes.

corrupt_value(*self*, *str*)

Method which corrupts the given input string and returns the modified string. See implementations in derived classes for details.

4.4 Class CorruptMissingValue

corruptor.CorruptValue
corruptor.CorruptMissingValue

```
A corruptor method which simply sets an attribute value to a missing
value.

The additional argument (besides the base class argument
'position_function') that has to be set when this attribute type is
initialised are:

missing_val  The string which designates a missing value. Default value
             is the empty string ''.

Note that the 'position_function' is not required by this corruptor
method.
```

4.4.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: corruptor.CorruptValue.__init__

corrupt_value(*self*, *in_str*)

Simply return the missing value string.

Overrides: corruptor.CorruptValue.corrupt_value

4.5 Class CorruptValueEdit

```
corruptor.CorruptValue ─┐
                        corruptor.CorruptValueEdit
```

```
A simple corruptor which applies one edit operation on the given value.

Depending upon the content of the value (letters, digits or mixed), if the
edit operation is an insert or substitution a character from the same set
(letters, digits or both) is selected.

The additional arguments (besides the base class argument
'position_function') that has to be set when this attribute type is
initialised are:

char_set_funct   A function which determines the set of characters that
                  can be inserted or used of substitution
insert_prob      These for values set the likelihood of which edit
delete_prob      operation will be selected.
substitute_prob  All four probability values must be between 0 and 1, and
transpose_prob   the sum of these four values must be 1.0
```

4.5.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: corruptor.CorruptValue.__init__

corrupt_value(*self*, *in_str*)

Method which corrupts the given input string and returns the modified string by randomly selecting an edit operation and position in the string where to apply this edit.

Overrides: corruptor.CorruptValue.corrupt_value

4.6 Class CorruptValueKeyboard

```
corruptor.CorruptValue ─┐
                        corruptor.CorruptValueKeyboard
```

```
Use a keyboard layout to simulate typing errors. They keyboard is
hard-coded into this method, but can be changed easily for different
keyboard layout.

A character from the original input string will be randomly chosen using
the position function, and then a character from either the same row or
column in the keyboard will be selected.
```

```
The additional arguments (besides the base class argument
'position_function') that have to be set when this attribute type is
initialised are:

row_prob  The probability that a neighbouring character in the same row
          is selected.

col_prob  The probability that a neighbouring character in the same
          column is selected.

The sum of row_prob and col_prob must be 1.0.
```

4.6.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: corruptor.CorruptValue.__init__

corrupt_value(*self*, *in_str*)

Method which corrupts the given input string by replacing a single character with a neighbouring character given the defined keyboard layout at a position randomly selected by the position function.

Overrides: corruptor.CorruptValue.corrupt_value

4.7 Class CorruptValueOCR

corruptor.CorruptValue

corruptor.CorruptValueOCR

```
Simulate OCR errors using a list of similar pairs of characters or strings
that will be applied on the original string values.

These pairs of characters will be loaded from a look-up file which is a
CSV file with two columns, the first is a single character or character
sequence, and the second column is also a single character or character
sequence. It is assumed that the second value is an OCR modification of
the first value, and the other way round. For example:

  5,S
  5,s
  2,Z
  2,z
  1,|
  6,G
```

```
It is possible for an 'original' string value (first column) to have
several variations (second column). In such a case one variation will be
randomly selected during the value corruption (modification) process.

The additional arguments (besides the base class argument
'position_function') that have to be set when this attribute type is
initialised are:

lookup_file_name  Name of the file which contains the OCR character
                   variations.

has_header_line   A flag, set to True or False, that has to be set
                   according to if the look-up file starts with a header
                   line or not.

unicode_encoding  The Unicode encoding (a string name) of the file.
```

4.7.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: corruptor.CorruptValue.__init__

corrupt_value(*self*, *in_str*)

Method which corrupts the given input string by replacing a single character or a sequence of characters with an OCR variation at a position randomly selected by the position function.

If there are several OCR variations then one will be randomly chosen.

Overrides: corruptor.CorruptValue.corrupt_value

4.8 Class CorruptValuePhonetic

corruptor.CorruptValue ─┐

corruptor.CorruptValuePhonetic

```
Simulate phonetic errors using a list of phonetic rules which are stored
in a CSV look-up file.

Each line (row) in the CSV file must consist of seven columns that contain
the following information:
1) Where a phonetic modification can be applied. Possible values are:
   'ALL','START','END','MIDDLE'
2) The original character sequence (i.e. the characters to be replaced)
3) The new character sequence (which will replace the original sequence)
4) Precondition: A condition that must occur before the original string
   character sequence in order for this rule to become applicable.
```

```
5) Postcondition: Similarly, a condition that must occur after the
   original string character sequence in order for this rule to become
   applicable.
6) Pattern existence condition: This condition requires that a certain
   given string character sequence does ('y' flag) or does not ('n' flag)
   occur in the input string.
7) Start existence condition: Similarly, this condition requires that the
   input string starts with a certain string pattern ('y' flag) or not
   ('n' flag)

A detailed description of this phonetic data generation is available in

  Accurate Synthetic Generation of Realistic Personal Information
  Peter Christen and Agus Pudjijono
  Proceedings of the Pacific-Asia Conference on Knowledge Discovery and
                     Data Mining (PAKDD), Bangkok, Thailand, April 2009.

For a given input string, one of the possible phonetic modifications will
be randomly selected without the use of the position function.

The additional arguments (besides the base class argument
'position_function') that have to be set when this attribute type is
initialised are:

lookup_file_name  Name of the file which contains the phonetic
                  modification patterns.

has_header_line   A flag, set to True or False, that has to be set
                  according to if the look-up file starts with a header
                  line or not.

unicode_encoding  The Unicode encoding (a string name) of the file.

Note that the 'position_function' is not required by this corruptor
method.
```

4.8.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: corruptor.CorruptValue.__init__

__apply_change__(*self*, *in_str*, *ch*)

Helper function which will apply the selected change to the input string.

Developed by Agus Pudjijono, ANU, 2008.

__slavo_germanic__(*self*, *in_str*)

Helper function which determines if the inputstring could contain a Slavo or Germanic name.

Developed by Agus Pudjijono, ANU, 2008.

__collect_replacement__(*self*, *s*, *where*, *orgpat*, *newpat*, *precond*, *postcond*, *existcond*, *startcond*)

```
Helper function which collects all the possible phonetic modification
patterns that are possible on the given input string, and replaces a
pattern in a string.

The following arguments are needed:
- where     Can be one of: 'ALL','START','END','MIDDLE'
- precond   Pre-condition (default 'None') can be 'V' for vowel or
            'C' for consonant
- postcond  Post-condition (default 'None') can be 'V' for vowel or
            'C' for consonant

Developed by Agus Pudjijono, ANU, 2008.
```

__get_transformation__(*self*, *in_str*)

Helper function which generates the list of possible phonetic modifications for the given input string.

Developed by Agus Pudjijono, ANU, 2008.

corrupt_value(*self*, *in_str*)

Method which corrupts the given input string by applying a phonetic modification.

If several such modifications are possible then one will be randomly selected.

Overrides: corruptor.CorruptValue.corrupt_value

4.9 Class CorruptCategoricalValue

corruptor.CorruptValue ─┐
corruptor.CorruptCategoricalValue

```
Replace a categorical value with another categorical value from the same
look-up file.

This corruptor can be used to modify attribute values with known
misspellings.

The look-up file is a CSV file with two columns, the first is a
categorical value that is expected to be in an attribute in an original
record, and the second is a variation of this categorical value.
```

```
It is possible for an 'original' categorical value (first column) to have
several misspelling variations (second column). In such a case one
misspelling will be randomly selected.

The additional arguments (besides the base class argument
'position_function') that have to be set when this attribute type is
initialised are:

lookup_file_name  Name of the file which contains the categorical values
                   and their misspellings.

has_header_line   A flag, set to True or False, that has to be set
                   according to if the look-up file starts with a header
                   line or not.

unicode_encoding  The Unicode encoding (a string name) of the file.

Note that the 'position_function' is not required by this corruptor
method.
```

4.9.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: corruptor.CorruptValue.__init__

corrupt_value(*self*, *in_str*)

Method which corrupts the given input string and replaces it with a misspelling, if there is a known misspelling for the given original value.

If there are several known misspellings for the given original value then one will be randomly selected.

Overrides: corruptor.CorruptValue.corrupt_value

4.10 Class CorruptDataSet

```
Class which provides methods to corrupt the original records generated by
one of the classes derived from the GenerateDataSet base class.

The following arguments need to be set when a GenerateDataSet instance is
initialised:

number_of_mod_records  The number of modified (corrupted) records that are
                        to be generated. This will correspond to the number
                        of 'duplicate' records that are generated.
```

```
number_of_org_records  The number of original records that were generated
                        by the GenerateDataSet class.

attribute_name_list     The list of attributes (fields) that have been
                        generated for each record.

max_num_dup_per_rec    The maximum number of modified (corrupted) records
                        that can be generated for a single original record.

num_dup_dist            The probability distribution used to create the
                        duplicate records for one original record (possible
                        distributions are: 'uniform', 'poisson', 'zipf')

max_num_mod_per_attr   The maximum number of modifications are to be
                        applied on a single attribute.

num_mod_per_rec         The number of modification that are to be applied
                        to a record

attr_mod_prob_dict      This dictionary contains probabilities that
                        determine how likely an attribute is selected for
                        random modification (corruption).
                        Keys are attribute names and values are probability
                        values. The sum of the given probabilities must sum
                        to 1.0.
                        Not all attributes need to be listed in this
                        dictionary, only the ones onto which modifications
                        are to be applied.
                        An example of such a dictionary is given below.

attr_mod_data_dict      A dictionary which contains for each attribute that
                        is to be modified a list which contains as pairs of
                        probabilities and corruptor objects (i.e. objects
                        based on any of the classes derived from base class
                        CorruptValue).
                        For each attribute listed, the sum of probabilities
                        given in its list must sum to 1.0.
                        An example of such a dictionary is given below.

Example for 'attr_mod_prob_dict':

attr_mod_prob_dict = {'surname':0.4, 'address':0.6}

In this example, the surname attribute will be selected for modification
with a 40% likelihood and the address attribute with a 60% likelihood.

Example for 'attr_mod_data_dict':

attr_mod_data_dict = {'surname':[(0.25,corrupt_ocr), (0.50:corrupt_edit),
                                 (0.25:corrupt_keyboard)],
                      'address':[(0.50:corrupt_ocr), (0.20:missing_value),
                                  (0.25:corrupt_keyboard)]}

In this example, if the 'surname' is selected for modification, with a
25% likelihood an OCR modification will be applied, with 50% likelihood a
character edit modification will be applied, and with 25% likelihood a
keyboard typing error modification will be applied.
If the 'address' attribute is selected, then with 50% likelihood an OCR
modification will be applied, with 20% likelihood a value will be set to
a missing value, and with 25% likelihood a keyboard typing error
modification will be applied.
```

4.10.1 Methods

__init__(*self*, ***kwargs*)

Constructor, set attributes.

corrupt_records(*self*, *rec_dict*)

Method to corrupt modify the records in the given record dictionary according to the settings of the data set corruptor.

5 Module generator

Module containing several classes to generate synthetic data according to user specification.

5.1 Variables

Name	Description
__package__	**Value:** `None`

5.2 Class GenerateAttribute

Known Subclasses: generator.GenerateFreqAttribute, generator.GenerateFuncAttribute

```
Base class for the definition of a single attribute (field) to be
generated.

This class and all of its derived classes provide methods that allow the
definition of a single attribute and the parameters necessary for its
generation.

The following variables need to be set when a GenerateAttribute instance
is initialised (with further parameters listed in the derived classes):

attribute_name  The name of this attribute, which will be used in the
                header line to be written into the output file.

                Ideally, this attribute name should be short, not contain
                spaces and it must not contain any quote or punctuation
                characters.
```

5.2.1 Methods

__init__(*self*, *base_kwargs*)

Constructor, set general attributes.

create_attribute_value(*self*)

Method which creates and returns one attribute value. See implementations in derived classes for details.

5.3 Class GenerateFreqAttribute

generator.GenerateAttribute
generator.GenerateFreqAttribute

```
Generate an attribute where values are retrieved from a lookup table that
contains categorical attribute values and their frequencies.

The additional argument (besides the base class argument 'attribute_name')
that has to be set when this attribute type is initialised are:

freq_file_name    The name of the file which contains the attribute values
                   and their frequencies.

                   This file must be in comma separated values (CSV) format
                   with the first column being the attribute values and the
                   second column their counts (positive integer numbers).

                   Each attribute value must only occur once in the
                   frequency file.

has_header_line   A flag, set to True or False, that has to be set
                   according to if the frequency file starts with a header
                   line or not.

unicode_encoding  The Unicode encoding (a string name) of the file.
```

5.3.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: generator.GenerateAttribute.__init__

create_attribute_value(*self*)

Method which creates and returns one attribute value randomly selected from the attribute value lookup table.

Overrides: generator.GenerateAttribute.create_attribute_value

5.4 Class GenerateFuncAttribute

```
generator.GenerateAttribute ─┐
                             generator.GenerateFuncAttribute
```

```
Generate an attribute where values are retrieved from a function that
creates values according to some specification.

Such functions include creating telephone numbers or social security
numbers with a certain structure, or numerical values normally or
uniformly distributed according to some parameter setting.

The additional argument (besides the base class argument 'attribute_name')
```

```
that has to be set when this attribute type is initialised are:

function    A Python function that, when called, has to return a string
            value that is created according to some specification.

parameters  A list of one or more parameters (maximum 5) passed to the
            function when it is called.
```

5.4.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: generator.GenerateAttribute.__init__

create_attribute_value(*self*)

Method which creates and returns one attribute value generated by the function provided.

Overrides: generator.GenerateAttribute.create_attribute_value

5.5 Class GenerateCompoundAttribute

Known Subclasses: generator.GenerateCateCateCompoundAttribute, generator.GenerateCateCateContCompoundAttribu
generator.GenerateCateContCompoundAttribute, generator.GenerateContContCompoundAttribute

Base class for the definition of compound attributes (fields) to be generated.

This class and all of its derived classes provide methods that allow the definition of several (are least two) attributes and the parameters necessary for their generation.

This base class does not have any generic variables that need to be set.

5.5.1 Methods

__init__(*self*, *base_kwargs*)

Constructor. See implementations in derived classes for details.

create_attribute_value(*self*)

Method which creates and returns several (compound) attribute values. See implementations in derived classes for details.

5.6 Class GenerateCateCateCompoundAttribute

generator.GenerateCompoundAttribute ─┐
　　generator.GenerateCateCateCompoundAttribute

```
Generate two attributes, both containing categorical values, where the
values of the second attribute depend upon the values in the first
attribute.

This for example allows the modelling of:
- city location values that depend upon gender values, or
- medication name values that depend upon gender values.

The arguments that have to be set when this attribute type is initialised
are:

categorical1_attribute_name  The name of the first categorical attribute
                              that will be generated. This name will be
                              used in the header line to be written into
                              the output file.

categorical2_attribute_name  The name of the second categorical attribute
                              that will be generated. This name will be
                              used in the header line to be written into
                              the output file.

lookup_file_name             Name of the file which contains the values of
                              the first categorical attribute, and for each
                              of these values the names of the categories
                              and their counts of the second categorical
                              attribute. This file format is further
                              explained below.

has_header_line              A flag, set to True or False, that has to be
                              set according to if the look-up file starts
                              with a header line or not.

unicode_encoding              The Unicode encoding (a string name) of the
                              file.

The format of the look-up file is:

# Comment lines start with the # character
cate_attr1_val,count,cate_attr2_val1,count1,cate_attr2_val2,count2,      cate_attr2_val3,count3,cate_attr2

The look-up file is a comma separated values (CSV) file which contains
two types of rows:
A) The first type of row contains the following columns:
   1) A categorical value. For all possible values of the first
      categorical attribute, one row must be specified in this look-up
      file.
   2) Count of this categorical value (a positive integer number). This
      determines the likelihood of how often a certain categorical value
      will be chosen. This count must be a positive integer number.
   3) The first categorical value of the second attribute.
   4) The count (positive integer number) of this first categorical
```

```
      value.
   5) The second categorical value of the second attribute.
   6) The count of this second categorical value.

   ...

   X) A '' character, which indicates that the following line (row)
      contains further categorical values and their counts from the
      second attribute.

B) The second type of row contains the following columns:
   1) A categorical value of the second attribute.
   2) The count of this categorical value.
   3) Another categorical value of the second attribute.
   4) The count of this categorical value.

   ...

Example:
  male,60,canberra,7,        sydney,30,melbourne,45,        perth,18
  female,40,canberra,10,sydney,40,        melbourne,20,brisbane,30,hobart,5,       perth,20
```

5.6.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: generator.GenerateCompoundAttribute.__init__

create_attribute_values(*self*)

Method which creates and returns two categorical attribute values, where the second value depends upon the first value. Both categorical values are randomly selected according to the provided frequency distributions.

Inherited from generator.GenerateCompoundAttribute(Section 5.5)

create_attribute_value()

5.7 Class GenerateCateContCompoundAttribute

```
generator.GenerateCompoundAttribute ─┐
                                     │
                                     generator.GenerateCateContCompoundAttribute
```

```
Generate two attributes, one containing categorical values and the other
continuous values, where the continuous values depend upon the categorical
values.
```

```
This for example allows the modelling of:
- salary values that depend upon gender values, or
- blood pressure values that depend upon age values.

The arguments that have to be set when this attribute type is initialised
are:

categorical_attribute_name  The name of the categorical attribute that
                             will be generated. This name will be used in
                             the header line to be written into the output
                             file.

continuous_attribute_name   The name of the continuous attribute that will
                             be generated. This name will be used in the
                             header line to be written into the output
                             file.

lookup_file_name            Name of the file which contains the values of
                             the continuous attribute, and for each of these
                             values the name of a function (and its
                             parameters) that is used to generate the
                             continuous values. This file format is further
                             explained below.

has_header_line             A flag, set to True or False, that has to be
                             set according to if the look-up file starts
                             with a header line or not.

unicode_encoding             The Unicode encoding (a string name) of the
                             file.

continuous_value_type       The format of how continuous values are
                             returned when they are generated. Possible
                             values are 'int', so integer values are
                             returned; or 'float1', 'float2', to 'float9',
                             in which case floating-point values with the
                             specified number of digits behind the comma
                             are returned.

The format of the look-up file is:

# Comment lines start with the # character
cate_val,count,funct_name,funct_param_1,...,funct_param_N
```

```
The look-up file is a comma separated values (CSV) file with the following
columns:
1) A categorical value. For all possible categorical values of an
   attribute, one row must be specified in this look-up file.

2) Count of this categorical value (a positive integer number). This
   determines the likelihood of how often a certain categorical value will
   be chosen.

3) A function which generates the continuous value for this categorical
   value. Implemented functions currently are:
   - uniform
   - normal

4) The parameters required for the function that generates the continuous
   values. They are:
   - uniform:  min_val, max_val
   - normal:   mu, sigma, min_val, max_val
               (min_val and max_val can be set to None in which case no
               minimum or maximum is enforced)

Example:
  male,60,uniform,20000,100000
  female,40,normal,35000,100000,10000,None
```

5.7.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: generator.GenerateCompoundAttribute.__init__

create_attribute_values(*self*)

Method which creates and returns two attribute values, one categorical and one continuous, with the categorical value randomly selected according to the provided frequency distribution, and the continuous value according to the selected function and its parameters.

Inherited from generator.GenerateCompoundAttribute(Section 5.5)

create_attribute_value()

5.8 Class GenerateCateCateContCompoundAttribute

generator.GenerateCompoundAttribute ─┐
generator.GenerateCateCateContCompoundAttribute

```
Generate three attributes, thefirst two containing categorical values and
the third containing continuous values, where the values of the second
attribute depend upon the values in the first attribute, and the values
of the third attribute depend upon both the values of the first and second
attribute.

This for example allows the modelling of:
- blood pressure depending upon gender and city of residence values, or
- salary depending upon gender and profession values.

The arguments that have to be set when this attribute type is initialised
are:

categorical1_attribute_name  The name of the first categorical attribute
                              that will be generated. This name will be
                              used in the header line to be written into
                              the output file.

categorical2_attribute_name  The name of the second categorical attribute
                              that will be generated. This name will be
                              used in the header line to be written into
                              the output file.

continuous_attribute_name    The name of the continuous attribute that
                              will be generated. This name will be used in
                              the header line to be written into the output
                              file.

lookup_file_name              Name of the file which contains the values
                               of the first categorical attribute, and for
                               each of these values the names of the
                               categories and their counts of the second
                               categorical attribute, and for each of these
                               values the name of a function (and its
                               parameters) that is used to generate the
                               continuous values. This file format is
                               further explained below.
```

has_header_line	A flag, set to True or False, that has to be set according to if the look-up file starts with a header line or not.
unicode_encoding	The Unicode encoding (a string name) of the file.
continuous_value_type	The format of how continuous values are returned when they are generated. Possible values are 'int', so integer values are returned; or 'float1', 'float2', to 'float9', in which case floating-point values with the specified number of digits behind the comma are returned.

```
The format of the look-up file is:

# Comment lines start with the # character
cate_attr1_val1,count
cate_attr2_val1,count,funct_name,funct_param_1,...,funct_param_N
cate_attr2_val2,count,funct_name,funct_param_1,...,funct_param_N
cate_attr2_val3,count,funct_name,funct_param_1,...,funct_param_N
...
cate_attr2_valX,count,funct_name,funct_param_1,...,funct_param_N
cate_attr1_val2,count
cate_attr2_val1,count,funct_name,funct_param_1,...,funct_param_N
cate_attr2_val2,count,funct_name,funct_param_1,...,funct_param_N
cate_attr2_val3,count,funct_name,funct_param_1,...,funct_param_N
...
cate_attr2_valX,count,funct_name,funct_param_1,...,funct_param_N
cate_attr1_val3,count
...


The look-up file is a comma separated values (CSV) file with the following
structure:

A) One row that contains two values:
   1) A categorical value of the first attribute. For all possible values
      of the first categorical attribute, one row must be specified in
      this look-up file.
   2) The count of this categorical value (a positive integer number).
      This determines the likelihood of how often a certain categorical
      value will be chosen.
```

```
B) After a row with two values, as described under A), one or more rows
   containing the following values in columns must be given:
   1) A categorical value from the second categorical attribute.
   2) The count of this categorical value (a positive integer number).
      This determines the likelihood of how often a certain categorical
      value will be chosen.
  3) A function which generates the continuous value for this categorical
     value. Implemented functions currently are:
     - uniform
     - normal
  4) The parameters required for the function that generates the
     continuous values. They are:
     - uniform:  min_val, max_val
     - normal:   mu, sigma, min_val, max_val
                 (min_val and max_val can be set to None in which case no
                 minimum or maximum is enforced)

Example:
  male,60
  canberra,20,uniform,50000,90000
  sydney,30,normal,75000,50000,20000,None
  melbourne,30,uniform,35000,200000
  perth,20,normal,55000,250000,15000,None
  female,40
  canberra,10,normal,45000,10000,None,150000
  sydney,40,uniform,60000,200000
  melbourne,20,uniform,50000,1750000
  brisbane,30,normal,55000,20000,20000,100000
```

5.8.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: generator.GenerateCompoundAttribute.__init__

create_attribute_values(*self*)

Method which creates and returns two categorical attribute values and one continuous value, where the second categorical value depends upon the first value, andthe continuous value depends on both categorical values. The two categorical values are randomly selected according to the provided frequency distributions, while the continuous value is generated according to the selected function and its parameters.

Inherited from generator.GenerateCompoundAttribute(Section 5.5)

create_attribute_value()

5.9 Class GenerateContContCompoundAttribute

generator.GenerateCompoundAttribute ─┐
generator.GenerateContContCompoundAttribute

```
Generate two continuous attribute values, where the value of the second
attribute depends upon the value of the first attribute.

This for example allows the modelling of:
- salary values that depend upon age values, or
- blood pressure values that depend upon age values.

The arguments that have to be set when this attribute type is initialised
are:

continuous1_attribute_name  The name of the first continuous attribute
                             that will be generated. This name will be
                             used in the header line to be written into
                             the output file.

continuous2_attribute_name  The name of the second continuous attribute
                             that will be generated. This name will be
                             used in the header line to be written into
                             the output file.

continuous1_funct_name      The name of the function that is used to
                             randomly generate the values of the first
                             attribute. Implemented functions currently
                             are:
                             - uniform
                             - normal
```

```
continuous1_funct_param     A list with the parameters required for the
                            function that generates the continuous values
                            in the first attribute. They are:
                            - uniform:  [min_val, max_val]
                            - normal:   [mu, sigma, min_val, max_val]
                                        (min_val and max_val can be set
                                        to None in which case no minimum
                                        or maximum is enforced)

continuous2_function        A Python function that has a floating-point
                            value as input (assumed to be a value
                            generated for the first attribute) and that
                            returns a floating-point value (assumed to be
                            the value of the second attribute).

continuous1_value_type      The format of how the continuous values in
                            the first attribute are returned when they
                            are generated. Possible values are 'int', so
                            integer values are generated; or 'float1',
                            'float2', to 'float9', in which case
                            floating-point values with the specified
                            number of digits behind the comma are
                            generated.

continuous2_value_type      The same as for the first attribute.
```

5.9.1 Methods

__init__(*self*, ***kwargs*)

Constructor. Process the derived keywords first, then call the base class constructor.

Overrides: generator.GenerateCompoundAttribute.__init__

create_attribute_values(*self*)

Method which creates and returns two continuous attribute values, with the the first continuous value according to the selected function and its parameters, and the second value depending upon the first value.

Inherited from generator.GenerateCompoundAttribute(Section 5.5)

create_attribute_value()

5.10 Class GenerateDataSet

```
Base class for data set generation.

This class and all of its derived classes provide methods that allow the
generation of a synthetic data set according to user specifications.

The following arguments need to be set when a GenerateDataSet instance is
initialised:

output_file_name     The name of the file that will be generated. This
                      will be a comma separated values (CSV) file. If the
                      file name given does not end with the extension
                      '.csv' then this extension will be added.

write_header_line    A flag (True or false) indicating if a header line
                      with the attribute (field) names is to be written at
                      the beginning of the output file or not. The default
                      for this argument is True.

rec_id_attr_name     The name of the record identifier attribute. This
                      name must be different from the names of all other
                      generated attributes. Record identifiers will be
                      unique values for each generated record.

number_of_records    The number of records that are to be generated. This
                      will correspond to the number of 'original' records
                      that are generated.

attribute_name_list  The list of attributes (fields) that are to be
                      generated for each record, and the sequence how they
                      are to be written into the output file. Each element
                      in this list must be an attribute name. These names
                      will become the header line of the output file (if
                      a header line is to be written).

attribute_data_list  A list which contains the actual attribute objects
                      (from the classes GenerateAttribute and
                      GenerateCompoundAttribute and their respective
                      derived classes).

unicode_encoding     The Unicode encoding (a string name) of the file.
```

5.10.1 Methods

__init__(*self*, ***kwargs*)

Constructor, set general attributes.

generate(*self*)

Method which runs the generation process and generates the specified number of records.

This method return a list containing the 'number_of_records' generated records, each being a dictionary with the keys being attribute names and values the corresponding attribute values.

write(*self*)

Write the generated records into the defined output file.

Index