Vertica ML Python Workshop

Exercise 1: Missing Values

Ouali Badr

December 5, 2019

Executive Summary

"Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world."

Louis Pasteur

VERTICA ML PYTHON allows the users to use Vertica advanced analytics and Machine Learning with a Python front-end Interface. In this exercise, you'll learn some basics to begin your fantastic Data Science Journey with the API. As a summary:

- Compute the columns count
- Drop missing values
- Impute missing values
- Drop columns

Contents

1 Presentation **3**

2 Functions used during the Exercise **3**

2.1 drop 3

2.2 dropna 4

2.3 fillna 5

3 Questions **6**

1 Presentation

Most of the time, the data will not been cleaned at the first data exploration. Missing values are one of the biggest problem in Data Preparation. Because of them, we can lack of different information or even worst they can lead to wrong assumptions. We can organize them into 3 categories:

- **MCAR:** Missing Completely at Random. The events that lead to any particular data-item being missing occur entirely at random. This type is not problematic as any mathematically correct imputation (median, average, mode...) will most of the time work without introducing bias in the data. For example, because of some transmission problems in IOT, some of the data related to the sensors could be lost.
- **MAR:** Missing at Random. It would have been better to call this type: Missing Conditionally at Random. The missingness is not random, but it is related to some of the observed data. Some of the data can easily explain the missing values. This situation needs a deep understanding of the data to choose the best way to impute the data. For example, some students may have not answered to some specific questions of a test because they were absent during the related lesson.
- **MNAR:** Missing not at Random. The value of the variable that's missing is related to the reason it?s missing. Most of the time the reason is known and the imputation is easy to make. For example, if someone didn't subscribe to a loyalty program we can leave an empty cell to avoid useless data storing.

During this exercise, we will find a way to impute the different features of the titanic dataset. This one has different types of missing values and it will give you different ideas to face this challenge.

2 Functions used during the Exercise

2.1 drop

Library: vertica_ml_python.vDataframe

Explanation: Drop the selected columns from the vDataframe. They will not be selected in the SQL statement.

```
vDataframe.drop(self, columns: list = [])
```

Drop the selected columns from the vDataframe.

Parameters

- **columns:** *<list>*, optional
  List of the vDataframe columns.

Returns

The vDataframe itself.

Example

```
from vertica_ml_python.vdataframe import vdf_from_relation
relation = "((SELECT True AS x, 4 AS y) UNION ALL (SELECT False AS x, 9 AS y))
    z"
```

```
vdf = vdf_from_relation(relation, dsn = "VerticaDSN")

#Output
          x     y
0      True     4
1     False     9
Name: VDF, Number of rows: 2, Number of columns: 2

vdf.drop(["x"])

#Output
     y
0    4
1    9
Name: y, Number of rows: 2, dtype: int
```

2.2 dropna

Library: vertica_ml_python.vDataframe

Explanation: Drop the selected columns missing values. A "WHERE" clause will be added in the SQL code generation.

```
vDataframe.dropna(self, columns: list = [])
```

Drop the vDataframe missing values.

Parameters

- **columns:** *<list>*, optional
  List of the vDataframe columns to consider. If this parameter is empty, it will filter all the rows having at least one missing element.

Returns

The vDataframe itself.

Example

```
from vertica_ml_python.vdataframe import vdf_from_relation
relation = "((SELECT 1 AS x, NULL AS y) UNION ALL (SELECT 1 AS x, 4 AS y)
    UNION ALL (SELECT 1 AS x, 5 AS y)) z"
vdf = vdf_from_relation(relation, dsn = "VerticaDSN")

#Output
     x        y
0    1     None
1    1        4
```

```
2     1        5
Name: VDF, Number of rows: 3, Number of columns: 2

vdf.dropna()

#Output
      x     y
0     1     4
1     1     5
Name: VDF, Number of rows: 2, Number of columns: 2
```

2.3 fillna

Library: vertica_ml_python.vDataframe[]

Explanation: Fill the column missing values. Many methods are available to give to any Data Scientists the freedom of choice. For personalised missing values filling, the 'eval' method (which will evaluate a Vertica expression) could be more suitable.

```
vDataframe[].fillna(
            self,
            val = None,
            method: str = "auto",
            by: list = [],
            order_by: list = [])
```

Fill the missing values using the input method. If the parameters val and method are empty, all the missing values will be filled automatically (using the average of the column for the numerical columns and the mode for the categorical ones)

Parameters

- **val:** *<anytype>*, optional
  Constant value used to impute the column.
- **method:** *<dict>*, optional
  Method used to impute the column.
  auto | avg | median | mode | ffill | backfill
  auto (default): average for numerical and mode for categorical.
  avg: Imputation using the column average.
  median: Imputation using the column median.
  mode: Imputation using the column mode (most occurrent element).
  ffill: Propagation of the first non-NULL element = constant interpolation (only for time series, `order_by` parameter must be defined).
  bfill: Back Propagation of the next non-NULL element = constant interpolation (only for time series, `order_by` parameter must be defined).
- **by:** *<list>*, optional
  The columns used to group the main one.

- **order_by:** *<list>*, optional
  The columns used to order the data (used only when method is bfill or ffill)

Returns

The parent vDataframe.

Example

```
from vertica_ml_python.vdataframe import vdf_from_relation
relation = "((SELECT 5 AS x) UNION ALL (SELECT NULL AS x) UNION ALL (SELECT 2
    AS x) UNION ALL (SELECT NULL AS x) UNION ALL (SELECT -2 AS x) UNION ALL (
    SELECT 12 AS x)) z"
vdf = vdf_from_relation(relation, dsn = "VerticaDSN")

#Output
        x
0       5
1    None
2       2
3    None
4      -2
5      12
Name: x, Number of rows: 6, dtype: int

vdf["x"].fillna()

#Output
         x
0     5.00
1     4.25
2     2.00
3     4.25
4    -2.00
5    12.00
Name: x, Number of rows: 6, dtype: float
```

3 Questions

Turn on Jupyter with the 'jupyter notebook' command. Start the notebook exercise1.ipynb and answer to the following questions.

- **Question 1:** Compute all the columns missing values using the count method. Define for each variable, each type of missing values we are facing.
- **Question 2:** By using the 'dropna' method, drop the elements having 1 or 2 missing elements.

- **Question 3:** The feature ’boat’ has many MNAR missing values. Indeed if someone paid for a lifeboat, he/she will get a lifeboat number otherwise this feature will be empty. By using the ’fillna’ method with the correct parameters, change the variable boat by returning 1 if the passenger paid for a lifeboat and 0 otherwise.
- **Question 4:** The column age has a lot of MCAR missing values. Imputing them using mathematical operation is very recommended as it will not add bias to the data. Use the ’fillna’ method to impute this column with the most precise way.
- **Question 5:** The column ’body’ is not at all suitable for the imputation as it represents the passengers ID. This ID is unique and there is no way to impute it. The column ’cabin’ represents the passengers cabin number. A lot of transformations must be made to extract information. Let’s drop them. Use the ’drop’ method to drop these columns.
- **Question 6:** The column ’home.dest’ is representing from where the passengers are embarking and to where they are going. It is a categorical variable. A possible to impute it is to use the mode (most occurrent element). Use the ’method’ fillna to impute it. You can then compute the number of missing elements per column to verify that all the data are well prepared.