Scientific Report – Data Cleaning Assignment

Machine Learning – Dr. Chen Hajaj, Faculty of Engineering, IEM
By: Nadav Erez

Introduction: In this assignment, we are to write a function that gets any dataset and returns it cleaned. The idea is that this is the first stage of cleaning that a dataset goes through in the whole process until it finally gets to the Data Scientist for analysis and deriving insights. In this report, I explain the methods I used for cleaning, and what limitations I've encountered in the process.

First, regarding **data exploration** – usually when cleaning a given dataset, we would like to apply functions such as:

data.head(), data.info(), data.shape, data.dtypes, sns.heatmap()

just so we can see how our data looks like, its size, how many features/rows, the feature types, how many NaN values etc.

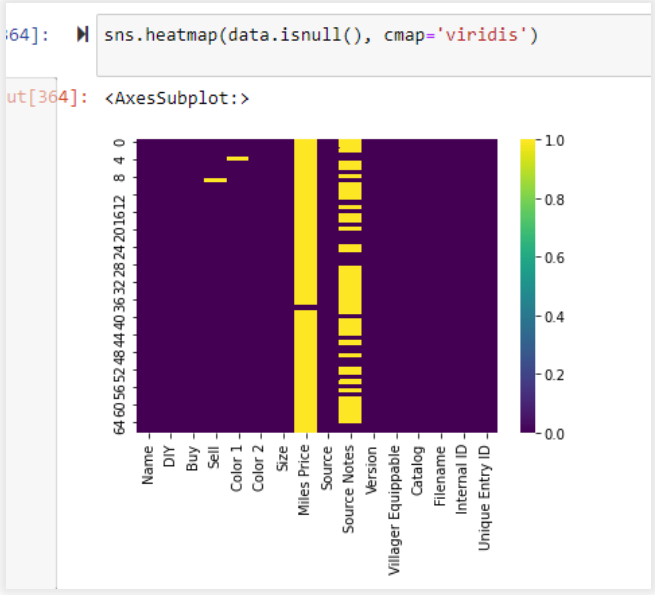

Figure 1: A heatmap showing us which features contain missing values

In this assignment, this visualization is less relevant, given that we just need a function that cleans the data, and it needs to apply to every dataset.

Standardization:

For strings that are in different format ('ISRAEL', Israel'): My strategy is unifying all to lower case (in the code: standard_lowerAndStrip()), and removing white spaces (In the case where a column containing names is changed to lower case letters, although it is not correct – it does not matter to the model).

Also, in the same function, I convert NaNs that appear in str type to np.nan so that we can identify them later as missing values as opposed to strings.

Limitations:

- Cleaning special characters (non-alphanumeric) is not possible if we don't know the data (@, #... could be relevant)
- Generally, when looking at the data itself you can see the instances where the data is different (19 and nineteen) and treat these cases individually.

Missing Values:

Usually when cleaning data and dealing with missing data (NaNs) we decide what to do based on each feature and the data itself.

In our case, we need to come up with a general approach for missing values. Methods used and limitations:

-Dropping duplicate rows after standardizing strings

-Dropping features and rows with NaN values:

There's no hard and fast rule to deciding at what threshold we can drop a feature from the data. By threshold I mean the minimum percentage of missing values – if a column has 10% null value – is that enough to drop the column?

I establish a general rule of dropping features containing 80% or more NaN values, and regarding rows – if the data has 50 rows or more – drop rows with 80% NaNs. The logic behind this is that features with so much missing data can be bad predictors for the response variable in the future model. Furthermore, I chose 50 as my threshold for dropping rows, however this is not ideal given that this can **introduce bias**. This was decided in lack of a better method.

Methods that are used in the industry include:

-Mean/Median Imputation – given a numeric column, replacing NaNs with mean of the column. Simple, but can be problematic, for example: a person might not mention his salary, thus there will be NaN in the Salary feature. If we impute it with the mean salary, we could overestimate this person's salary (he was not comfortable giving his salary due to it being lower than average) thus introducing bias. Specifically in this example, because we are talking about salary, we could use the median salary as a better method for replacing missing values. But our solution needs to apply to every dataset given. Another problem with mean imputation: if a feature has a high percentage of missing values, replacing them with the mean will lower the variance of the feature.

Other methods include regression imputation, hot & cold imputation, stochastic imputation.

Although not perfect, I decided on going with **mean imputation** for **float** features with missing values below 10%, by using my mean_for_float function.

-mean_for_float(feature, data) – computes the mean but only considers float values, and disregards strings/nulls etc.

I only do this on float type features and not integers because we can't know from a generic dataset what the primary key is. Because of this, replacing NaNs in numeric features is not possible because a feature can be defined as numeric, but actually it contains IDs, or passport numbers (just a few examples).

Ideally, I would have liked to use regression imputation for missing values, at least in numeric columns, as this can use prediction to replace missing values and can be more accurate.

In the code, my intention is to classify the features by their **most frequent data type.** For this, I constructed two functions:

- get_str_real_type(item_) – this function gets an item from the feature in the pandas dataframe and returns its "real" data type (e.g if the item is '9.0' – the function will return float).

- most_frequent_type(df, col_name) – this function gets a dataframe and a feature and returns the feature's most frequent data type. If we know the most frequent data type, we can replace the other values.

Other functions:

-missing_values(data,freq_types_dict) – this function drops rows and columns based on percentage, as mentioned above.

-correlation(data) – this function drops features that are highly correlated (>0.95).

Other limitations:

-There's no way of knowing if a string-type categorical feature is nominal or ordinal.

-Some features can't contain negative values, like Age, Salary, and such. Usually we could use df.describe to see the minimum value of each feature. If there's a negative value in features that can't be negative, we can replace these values with a 0, the mean or just drop these rows, depending on the data. So it would look like this:

```
for i in range(df.shape[0]):
    if df.Age.iloc[i] < 0.0:
        df.Age.iloc[i] = 20.0
```

Figure 2: Replacing negative values by different methods

Because we are writing a general code, there is no option to know if a certain feature can be negative or not.