Lecturer:
Prof. Dr. Ajinkya Prabhune

Student:
Elnaz Dehkharghani

Date:
18-03-2021

Matrikelnummer:
11015404

Data Management 1

Goal B:
Uncleaning the Berlin Airbnb Dataset

In this project for both Goal A and B one dataset as Berlin Airbnb data is considered. This dataset is downloaded from Kaggle.com which is scraped on 07th, 2018, and contains detailed listings data, review data, and calendar data. In the zip file, there is a folder named Datasets which contains three initial downloaded datasets.

As the number of datasets and columns is too much, before starting my project I have defined eight different user stories that could help me to work on the desired number of the columns and datasets. The user stories and the result of them with the visualization part are completely explained in the presentation of Goal A. The project steps are as follows:

First: Making an unclean dataset with Pandas for Goal B in the Jupyter Notebook.
Second: Using the uncleaned dataset from Goal B for Goal A with two different approaches:

- Data Profiling - in both Pandas and Talend Data Quality Management tool.
- Data Cleaning - in OpenRefine: The uncleaned dataset from Goal B is completely cleaned in OpenRefine.
- Data Cleaning - in Jupyter Notebook, Panda's library: Filling the null values of the partially cleaned dataset by OpenRefine with the help of the original values of the downloaded dataset from Kaggle.com then visualizing it.

Preparing the dataset for Goal B:

The project is started with importing the required libraries. Then two different datasets from the Datasets folder are importing into the Jupyter notebook. All the 15 columns in the listings.csv are exactly same by columns of listings_summary.csv which has 96 columns. But I for the inconsistency purpose need to join these two datasets together. At the initial insight, there are no duplicates in both of them. In this step, just one column as last_review is selected from the listings.csv dataset and then left joined to the listings_summary.csv. The result is shown by head () function as merged_df with 22552 rows and 97 columns. Since the number of columns for the aim of this project is too much, with the help of my predefined user stories and covering the exam purpose, I could select 26 required columns of this merged dataset. Also, there was not any possibility to create a new column with the gender (I could not find any Mr., Ms. and Mrs. before names to split columns base on them and then make a gender column) and since it was a must to have a char datatype column, I had to generate a new column with fake genders which has an inaccurate entities with four different probabilities of F, M, C, and G. The dataset is now completely created and it is ready for working on Goal B. This dataset is named “firstDataset_Raw.csv”.

Goal B:

Goal B is started with importing the “firstDataset_Raw.csv”. It has 22,552 rows and 27 columns in total. I have to unclean this dataset for at least 25%. I have decided to do this uncleaning on the 25% of the cells which will make the dataset more unclean and messier. In this way, even more than 25% my records (5,638) will be uncleaned that in further we will see and calculate its amount. For this purpose, I calculated the total number of the cells as 608,904, then found 25% of it as 152,226. This amount of uncleaning should be considered on all seven data quality dimensions which are as follows:

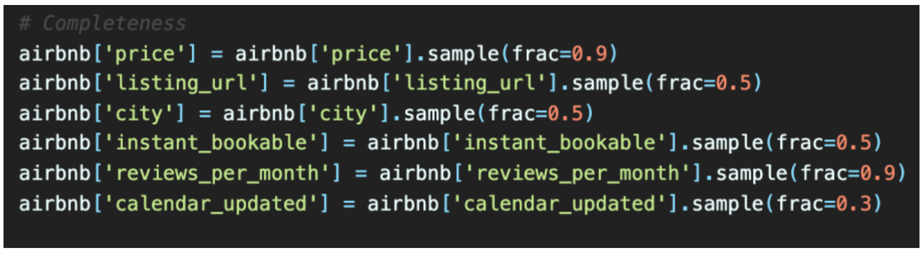
1. Accuracy
2. Completeness
3. Conformity
4. Validity
5. Consistency
6. Uniqueness
7. Currency

Completeness:

To achieve this goal and making some cells blank or null, six of the columns are selected, but before applying the incompleteness operation we will have a look at these columns if they have already null values or not:

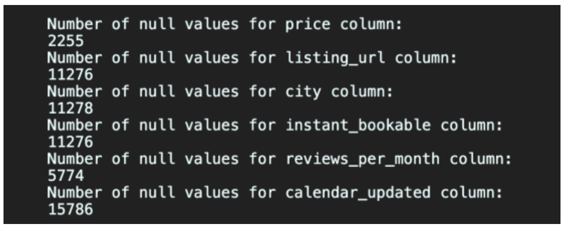
- **Price:** fully complete
- **Listing_url:** fully complete
- **City:** 5 null values
- **Instant_bookable:** fully complete
- **Reviews_per_month:** 3914 null values
- **Calender_updated:** fully complete

It is easy to apply incompleteness to these columns just by making some fractions of sampling, the remaining cells will be with null values.



```
# Completeness
airbnb['price'] = airbnb['price'].sample(frac=0.9)
airbnb['listing_url'] = airbnb['listing_url'].sample(frac=0.5)
airbnb['city'] = airbnb['city'].sample(frac=0.5)
airbnb['instant_bookable'] = airbnb['instant_bookable'].sample(frac=0.5)
airbnb['reviews_per_month'] = airbnb['reviews_per_month'].sample(frac=0.9)
airbnb['calendar_updated'] = airbnb['calendar_updated'].sample(frac=0.3)
```

The result is like this:

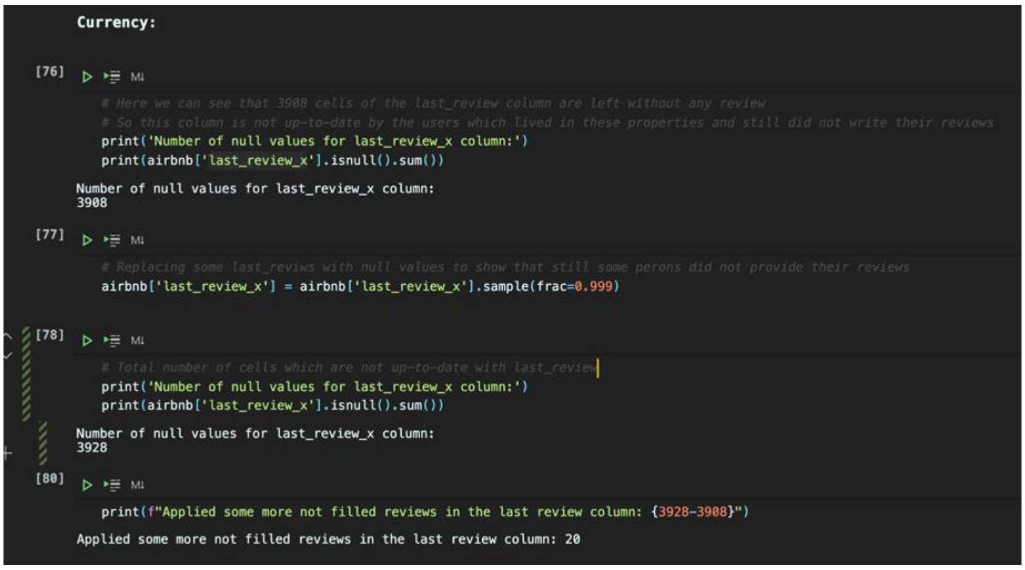

```
Number of null values for price column:
2255
Number of null values for listing_url column:
11276
Number of null values for city column:
11278
Number of null values for instant_bookable column:
11276
Number of null values for reviews_per_month column:
5774
Number of null values for calendar_updated column:
15786
```

In this step with calculating the initial null values, I have *53,726* incomplete cells in total. Here it is obvious that this incompleteness is applied on more than 25% rows which is 5,638.

Currency:

Just by applying the incompleteness dimension in the previous step I could actually cover the currency dimension for the following columns, but the first and last review columns and the number of reviews are more likely to have currency problem:

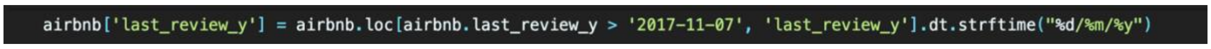
- **First_review, Last_review:** In these columns we have actually have currency problem since 3908 cells are empty, but I for covering the currency problem applied again more 20 cells to make nulls for the column last_review. These are those listings which are new or people have already lived in them but still did not provide any reviews and maybe in the future they will add their reviews there, so this column is not up-to-date by our users.
- **Number_of_reviews:** Still some users did not write their reviews, it is not up-to-date.
- **Price:** Price is null and we do not have access to the up-to-date values.
- **Reviews_per_month:** It is pending with some of the users which have already stayed in the properties but left it without any reviews.
- **Calender_updated:** If it is null, we do not know how the information provided by the host is up-to-date and even reliable for further analysis.


Currency:

```
[76]
# Here we can see that 3908 cells of the last_review column are left without any review
# So this column is not up-to-date by the users which lived in these properties and still did not write their reviews
print('Number of null values for last_review_x column:')
print(airbnb['last_review_x'].isnull().sum())

Number of null values for last_review_x column:
3908

[77]
# Replacing some last_reviws with null values to show that still some perons did not provide their reviews
airbnb['last_review_x'] = airbnb['last_review_x'].sample(frac=0.999)

[78]
# Total number of cells which are not up-to-date with last_review
print('Number of null values for last_review_x column:')
print(airbnb['last_review_x'].isnull().sum())

Number of null values for last_review_x column:
3928

[80]
print(f"Applied some more not filled reviews in the last review column: {3928-3908}")

Applied some more not filled reviews in the last review column: 20
```

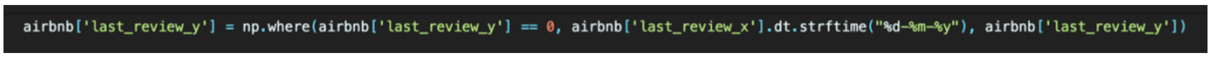
Consistency:

I have already left-joined the last_review_y column, it is time for making inconsistency. In this column, we have 3908 null values (currency and completeness problem) and time format is like yyyy-mm-dd. With the following code I made inconsistency for years after 2017-11-07 with the time format of d/m/y.



```
airbnb['last_review_y'] = airbnb.loc[airbnb.last_review_y > '2017-11-07', 'last_review_y'].dt.strftime("%d/%m/%y")
```

The number of null values increase to 8903. We change the null values with 0, after that replace these zeros with the original values from the last_review_x and at same time change time format of last_review_x to d-m-y, and if we do not have any value for last_review_y it will remain as before.



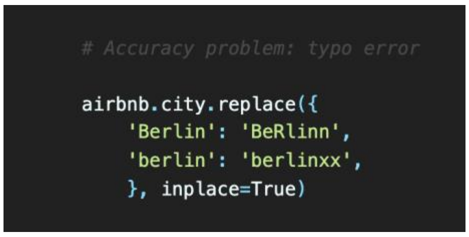
```
airbnb['last_review_y'] = np.where(airbnb['last_review_y'] == 0, airbnb['last_review_x'].dt.strftime("%d-%m-%y"), airbnb['last_review_y'])
```

Since in this step I have changed all the cells value with two different type of date format the number of changing cells or records are: *18,640* in total which is again more than 25% rows calculated as 5,638.
Now in our dataset, there is an inconsistent column with two different date formats inside itself and also in comparison with two other existed date columns that are first_review and host_since, which their date format is like yyyy-mm-dd.

Accuracy:

A. Typo error on city column:

We have a column with the name of city. Since we are working on the Berlin Airbnb listings, we know that the only possible value for this column is Berlin. As we can see there are already too many irrelevant values in this column but to make more inaccuracy, I have replaced Berlins and berlins in it with some other words with typo errors.

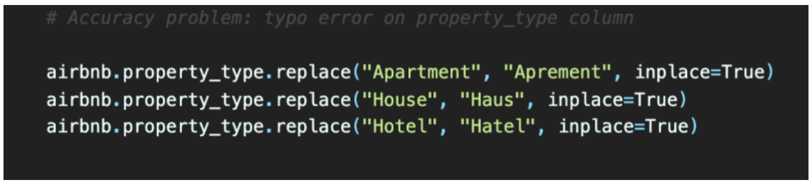


```
# Accuracy problem: typo error

airbnb.city.replace({
    'Berlin': 'BeRlinn',
    'berlin': 'berlinxx',
    }, inplace=True)
```

In this step *11,216* cells or records have been changed in total, again it is more than 25% rows calculated as 5,638.

SRH HOCHSCHULE HEIDELBERG

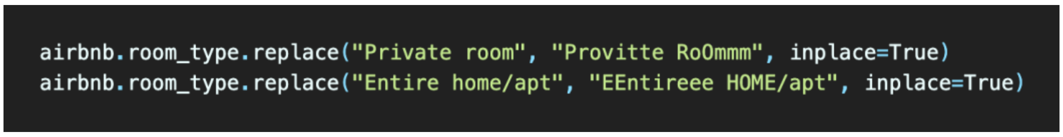
B. Typo error on property_type column:



```
# Accuracy problem: typo error on property_type column

airbnb.property_type.replace("Apartment", "Aprement", inplace=True)
airbnb.property_type.replace("House", "Haus", inplace=True)
airbnb.property_type.replace("Hotel", "Hatel", inplace=True)
```

In this step 20,673 cells have been changed in total.

C. Typo error on room_type column:



```
airbnb.room_type.replace("Private room", "Provitte RoOmmm", inplace=True)
airbnb.room_type.replace("Entire home/apt", "EEntireee HOME/apt", inplace=True)
```

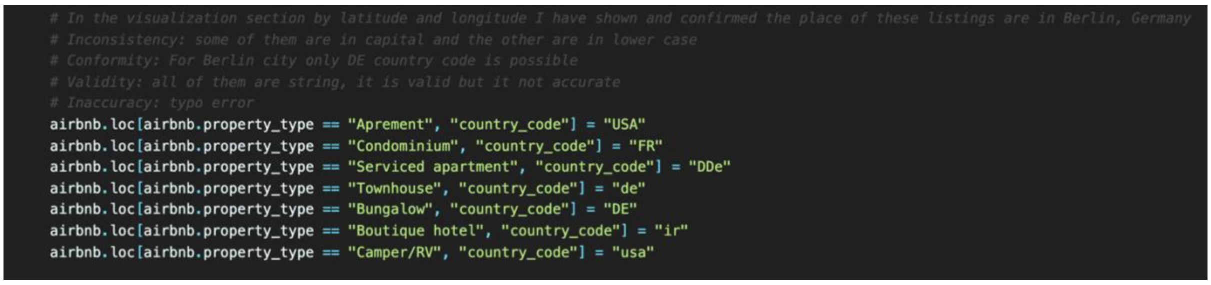
In this step 22,256 cells have been changed in total.

Conformity, Consistency, Accuracy:

A. Country_code

Here I am aiming at the same time cover three different quality dimension problems in just one column as country_code. So, I have filtered this column base on different types of the properties and then applied values with inconsistency and typo errors in it. The conformity quality dimension problem is covered since the only possible value for this column would be DE. Having the city column as Berlin and checking it with external source to confirm that DE is the only possible country code for this column.

- In the visualization section by latitude and longitude I have shown and confirmed the place of these listings are in Berlin, Germany.



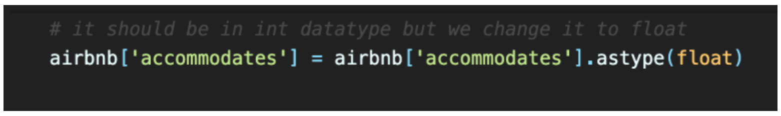
```
# In the visualization section by latitude and longitude I have shown and confirmed the place of these listings are in Berlin, Germany
# Inconsistency: some of them are in capital and the other are in lower case
# Conformity: For Berlin city only DE country code is possible
# Validity: all of them are string, it is valid but it not accurate
# Inaccuracy: typo error
airbnb.loc[airbnb.property_type == "Aprement", "country_code"] = "USA"
airbnb.loc[airbnb.property_type == "Condominium", "country_code"] = "FR"
airbnb.loc[airbnb.property_type == "Serviced apartment", "country_code"] = "DDe"
airbnb.loc[airbnb.property_type == "Townhouse", "country_code"] = "de"
airbnb.loc[airbnb.property_type == "Bungalow", "country_code"] = "DE"
airbnb.loc[airbnb.property_type == "Boutique hotel", "country_code"] = "ir"
airbnb.loc[airbnb.property_type == "Camper/RV", "country_code"] = "usa"
```

In this step 21,165 cells have been changed in total.

Conformity, Validity, Accuracy:

A. Accommodates column

In this column I have changed the datatype to float to cover the conformity, validity, accuracy problem. It is not valid base on our business rules and also accurate to have float numbers in this column and with conformity check with an external source we know that the only possible datatype is int.



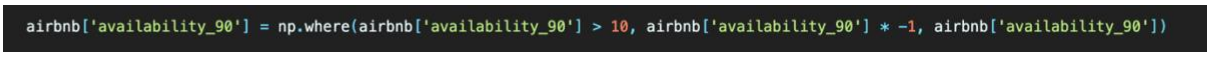
```
# it should be in int datatype but we change it to float
airbnb['accommodates'] = airbnb['accommodates'].astype(float)
```

In this step *22,552* cells have been changed in total.

Validity:

A. Availability_90:

Setting some values as negative. The only possible values base on the business rules for the column availibity_90 is to have a positive and less than 90 days. If everything violet these rules then it would be invalid.

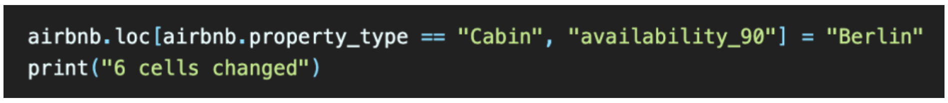


```
airbnb['availability_90'] = np.where(airbnb['availability_90'] > 10, airbnb['availability_90'] * -1, airbnb['availability_90'])
```

In this step *8,790* cells have been changed in total.

B. Availability_90:

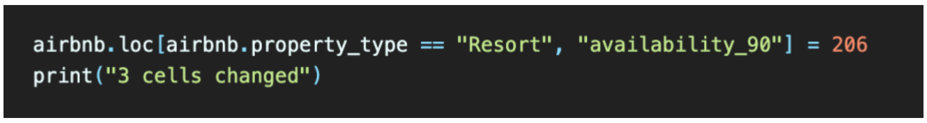
Inserting Berlin string into this column. It is not valid to have string in this column.



```
airbnb.loc[airbnb.property_type == "Cabin", "availability_90"] = "Berlin"
print("6 cells changed")
```

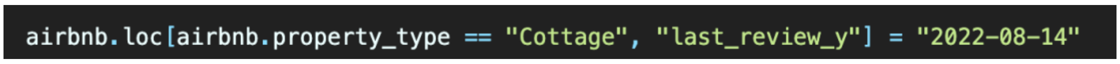
C. Availability_90:

Inserting 206 into this column can be consider as outlier.



```
airbnb.loc[airbnb.property_type == "Resort", "availability_90"] = 206
print("3 cells changed")
```

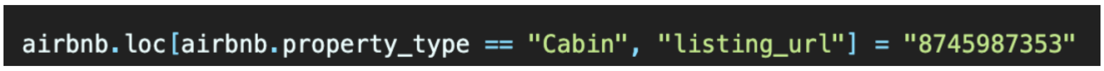
D. Last_review_y:



```
airbnb.loc[airbnb.property_type == "Cottage", "last_review_y"] = "2022-08-14"
```

Setting one out of range date time in the column last_review_y. Here only four columns has been changed and since I have already considered number of changing cells for this column I do not count this four values as messy values.

E. Listings_url:

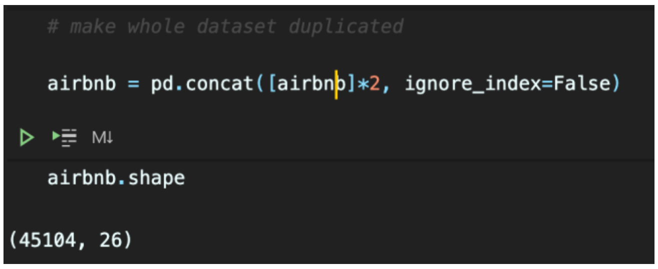


```
airbnb.loc[airbnb.property_type == "Cabin", "listing_url"] = "8745987353"
```

Inserting one invalid value in the column listing_url. In this step 6 cells have been changed in total.

Uniqueness:

Here I made for each of the rows one duplicate. So, this is applied for whole number of rows which are 22552 and we would have 50% duplicates in our dataset.



```
# make whole dataset duplicated

airbnb = pd.concat([airbnb]*2, ignore_index=False)
```

```
airbnb.shape
```

```
(45104, 26)
```

Calculating the total number of cells which are now messy:

6 + 8790 + 22552 + 21165 + 22256 + 20673 + 11209 + 18644 + 53726 = 179,024

- Around 29% of the cells without considering the 50% duplicated rows is now messy. As we have seen before if I wanted to apply this 25% on rows, I only needed to make 5638 rows messy but I only for some of the quality dimensions did this uncleaning on whole records, so as record view it has around 98% unclean rows.

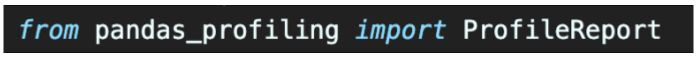
The unclean dataset with the name of: Uncleaned_GoalB.csv will be used for the cleaning part in Goal A with OpenRefine.

End of Goal B

Goal A:
Data Profiling with Pandas
(Second way)

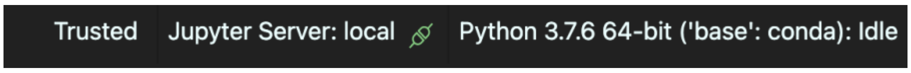
In this step I would like to apart from data profiling in the talend data quality management tool, do this profiling also in the pandas:

For this purpose, we have to import the following library:



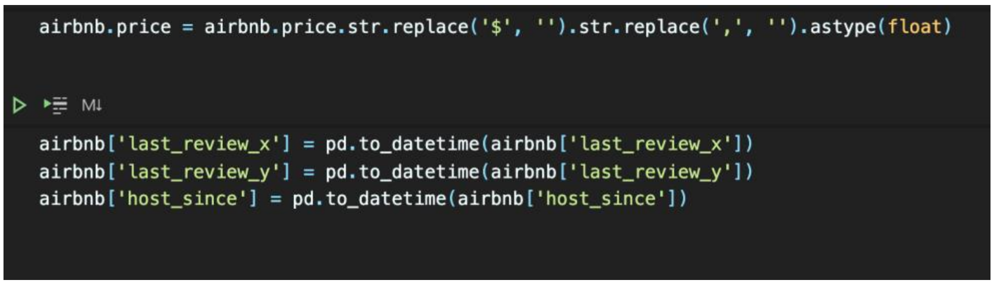
```
from pandas_profiling import ProfileReport
```

And set the environment to Conda:



Trusted | Jupyter Server: local | Python 3.7.6 64-bit ('base': conda): Idle

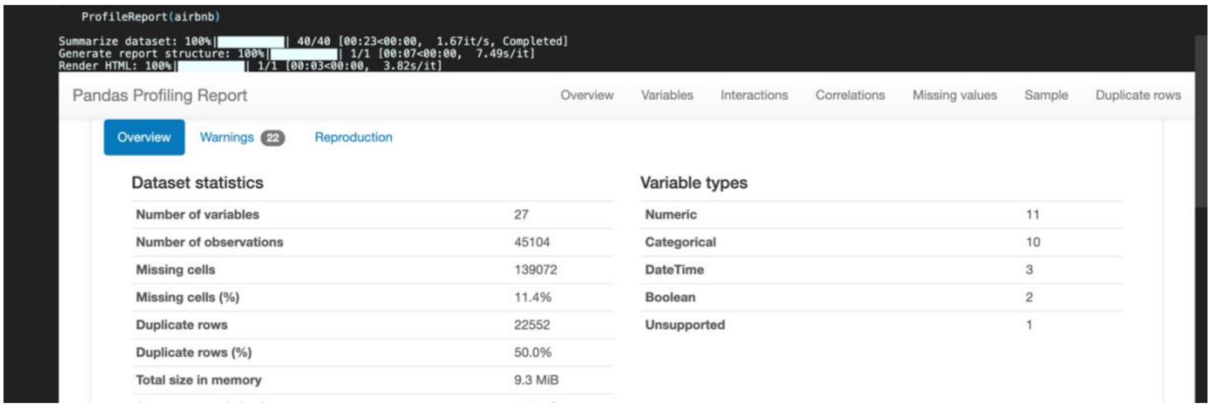
Then we have to change some column datatypes to its correct format and do some data cleaning on the price column to consider it as a numeric column.



```
airbnb.price = airbnb.price.str.replace('$', '').str.replace(',', '').astype(float)
```

```
airbnb['last_review_x'] = pd.to_datetime(airbnb['last_review_x'])
airbnb['last_review_y'] = pd.to_datetime(airbnb['last_review_y'])
airbnb['host_since'] = pd.to_datetime(airbnb['host_since'])
```

Now, with single line of code we can profile our data into a new rendered HTML page:



```
ProfileReport(airbnb)
Summarize dataset: 100%|██████████| 40/40 [00:23<00:00,  1.67it/s, Completed]
Generate report structure: 100%|██████████| 1/1 [00:07<00:00,  7.49s/it]
Render HTML: 100%|██████████| 1/1 [00:03<00:00,  3.82s/it]
```

Dataset statistics

Number of variables	27
Number of observations	45104
Missing cells	139072
Missing cells (%)	11.4%
Duplicate rows	22552
Duplicate rows (%)	50.0%
Total size in memory	9.3 MiB

Variable types

Numeric	11
Categorical	10
DateTime	3
Boolean	2
Unsupported	1

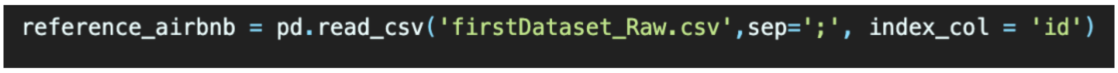
The explanation of data profiling is covered in presentation.

Goal A:
Data Cleaning with Pandas
(Second approach)

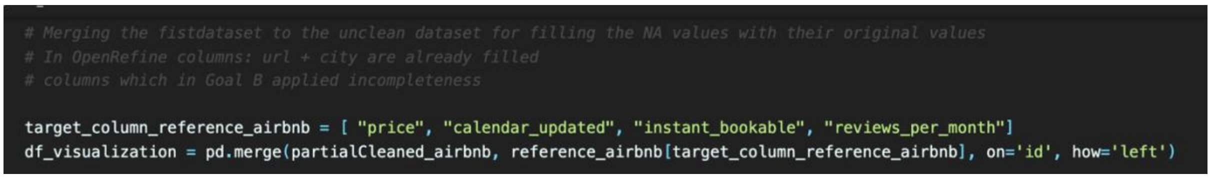
For covering my second approach in data cleaning which is filling the null values with their original values where they have been removed during the uncleaning section in Goal B, I import my partially cleaned dataset with OpenRefine into my Jupyter notebook "partialCleaned_Continue_Cleaning_in_Pandas.csv". Since for these null values I have already access to their original one, it was more interesting for me to have the exact insight in my visualization section especially for the price column which is my important column in future prediction and modeling.

- Also, I have completed my cleaning in OpenRefine and its recipe and complete cleaned dataset is provided in the zip file. The only difference between cleaning in pandas and OpenRefine is filling the null values. In the OpenRefine the null values are filled with the Mean values.

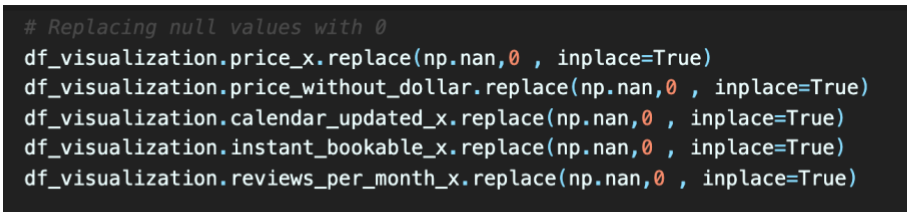
There was a simple way to fill the listings_url column and City column with OpenRefine. Here we only deal with the remaining columns which have null values.



```
reference_airbnb = pd.read_csv('firstDataset_Raw.csv',sep=';', index_col = 'id')
```

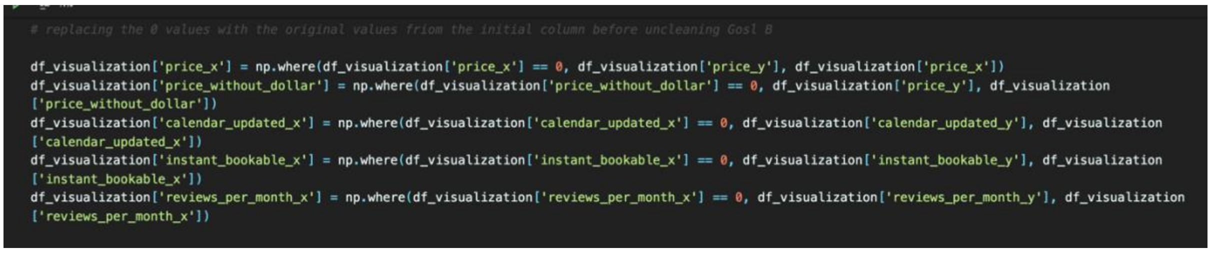


```
# Merging the fistdataset to the unclean dataset for filling the NA values with their original values
# In OpenRefine columns: url + city are already filled
# columns which in Goal B applied incompleteness

target_column_reference_airbnb = [ "price", "calendar_updated", "instant_bookable", "reviews_per_month"]
df_visualization = pd.merge(partialCleaned_airbnb, reference_airbnb[target_column_reference_airbnb], on='id', how='left')
```



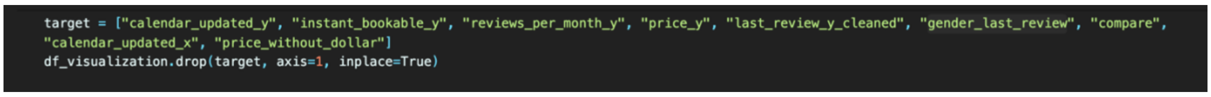
```
# Replacing null values with 0
df_visualization.price_x.replace(np.nan,0 , inplace=True)
df_visualization.price_without_dollar.replace(np.nan,0 , inplace=True)
df_visualization.calendar_updated_x.replace(np.nan,0 , inplace=True)
df_visualization.instant_bookable_x.replace(np.nan,0 , inplace=True)
df_visualization.reviews_per_month_x.replace(np.nan,0 , inplace=True)
```

Completeness: Now all the null values are filled with their original and exact values.



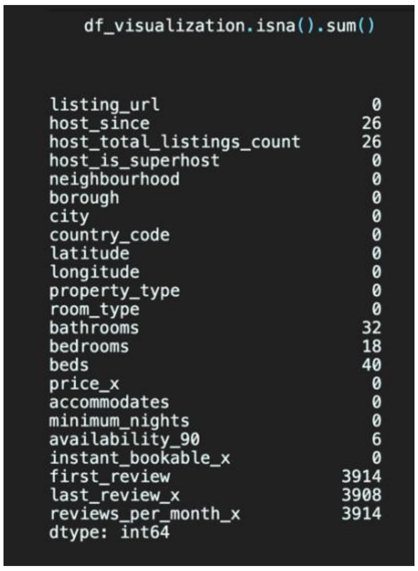
```
# replacing the 0 values with the original values friom the initial column before uncleaning Gosl B

df_visualization['price_x'] = np.where(df_visualization['price_x'] == 0, df_visualization['price_y'], df_visualization['price_x'])
df_visualization['price_without_dollar'] = np.where(df_visualization['price_without_dollar'] == 0, df_visualization['price_y'], df_visualization
['price_without_dollar'])
df_visualization['calendar_updated_x'] = np.where(df_visualization['calendar_updated_x'] == 0, df_visualization['calendar_updated_y'], df_visualization
['calendar_updated_x'])
df_visualization['instant_bookable_x'] = np.where(df_visualization['instant_bookable_x'] == 0, df_visualization['instant_bookable_y'], df_visualization
['instant_bookable_x'])
df_visualization['reviews_per_month_x'] = np.where(df_visualization['reviews_per_month_x'] == 0, df_visualization['reviews_per_month_y'], df_visualization
['reviews_per_month_x'])
```

For covering the user story and visualizing section I removed the unnecessary and repeated columns:



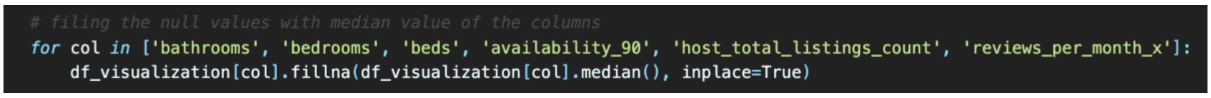
```
target = ["calendar_updated_y", "instant_bookable_y", "reviews_per_month_y", "price_y", "last_review_y_cleaned", "gender_last_review", "compare",
"calendar_updated_x", "price_without_dollar"]
df_visualization.drop(target, axis=1, inplace=True)
```

These are the remaining columns which I have to deal to fill them.

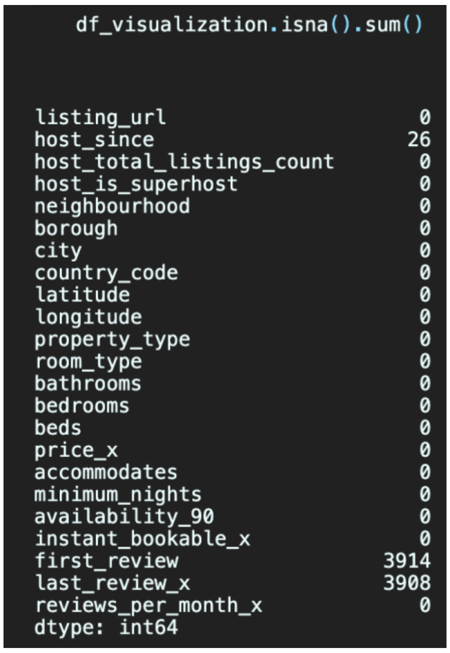


```
df_visualization.isna().sum()


listing_url                      0
host_since                      26
host_total_listings_count       26
host_is_superhost                0
neighbourhood                    0
borough                          0
city                             0
country_code                     0
latitude                         0
longitude                        0
property_type                    0
room_type                        0
bathrooms                       32
bedrooms                        18
beds                            40
price_x                          0
accommodates                     0
minimum_nights                   0
availability_90                  6
instant_bookable_x               0
first_review                  3914
last_review_x                 3908
reviews_per_month_x           3914
dtype: int64
```

These columns are filled with the median values:



```
# filing the null values with median value of the columns
for col in ['bathrooms', 'bedrooms', 'beds', 'availability_90', 'host_total_listings_count', 'reviews_per_month_x']:
    df_visualization[col].fillna(df_visualization[col].median(), inplace=True)
```

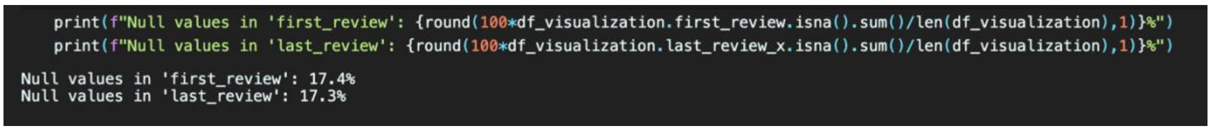


```
df_visualization.isna().sum()


listing_url                      0
host_since                      26
host_total_listings_count        0
host_is_superhost                0
neighbourhood                    0
borough                          0
city                             0
country_code                     0
latitude                         0
longitude                        0
property_type                    0
room_type                        0
bathrooms                        0
bedrooms                         0
beds                             0
price_x                          0
accommodates                     0
minimum_nights                   0
availability_90                  0
instant_bookable_x               0
first_review                  3914
last_review_x                 3908
reviews_per_month_x              0
dtype: int64
```

first_review and last_review:

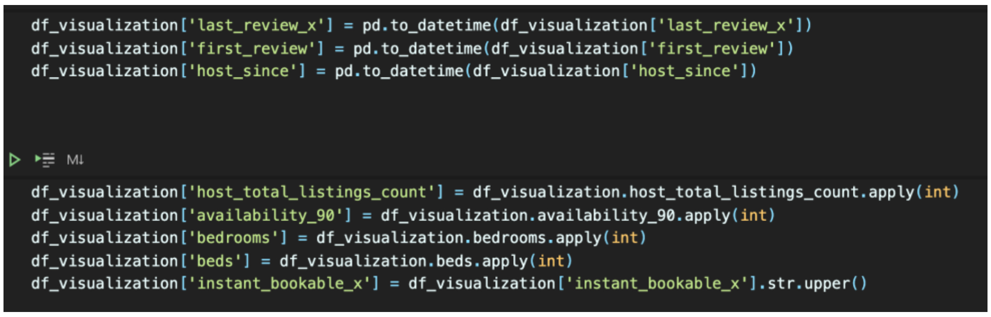
About 17% of listings have not had a review written for them. This is a bit large proportion of the dataset to drop since dropping the columns would lose a lot of useful information. Reviews play an important role in people's decisions to book, and therefore in price. Also, it is large proportion to simply replace with median or mean values, as this would skew the distribution substantially.

These missing values here are not really missing values, it tells us that these are new or previously unbooked listings that have not had reviews yet.
In order to make the resulting model work able to predict prices for any Airbnb listing, including brand new listings, it is actually beneficial to keep them in. Therefore, these will be kept as an 'unknown' category, and the feature will have to be treated as categorical (and therefore one-hot encoded) rather than numerical.



```
    print(f"Null values in 'first_review': {round(100*df_visualization.first_review.isna().sum()/len(df_visualization),1)}%")
    print(f"Null values in 'last_review': {round(100*df_visualization.last_review_x.isna().sum()/len(df_visualization),1)}%")

Null values in 'first_review': 17.4%
Null values in 'last_review': 17.3%
```

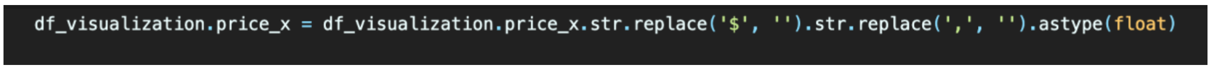
In continue for more cleaning purpose, change some of the column’s datatypes:



```
df_visualization['last_review_x'] = pd.to_datetime(df_visualization['last_review_x'])
df_visualization['first_review'] = pd.to_datetime(df_visualization['first_review'])
df_visualization['host_since'] = pd.to_datetime(df_visualization['host_since'])
```

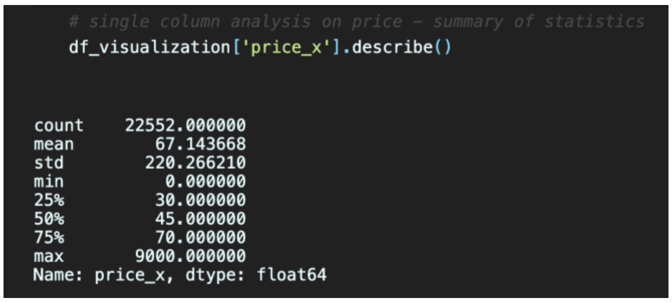
```
df_visualization['host_total_listings_count'] = df_visualization.host_total_listings_count.apply(int)
df_visualization['availability_90'] = df_visualization.availability_90.apply(int)
df_visualization['bedrooms'] = df_visualization.bedrooms.apply(int)
df_visualization['beds'] = df_visualization.beds.apply(int)
df_visualization['instant_bookable_x'] = df_visualization['instant_bookable_x'].str.upper()
```

And do cleaning on the price column:

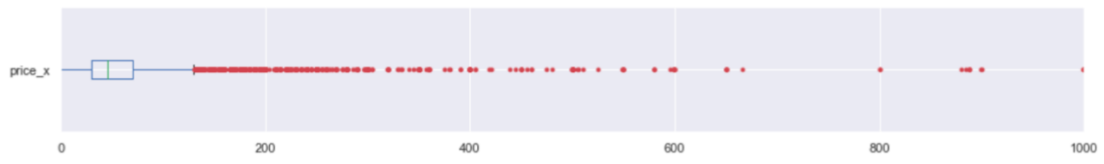


```
df_visualization.price_x = df_visualization.price_x.str.replace('$', '').str.replace(',', '').astype(float)
```

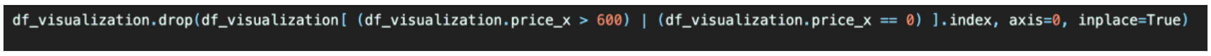
Here, I have found some outliers in price column which is related to validity dimension problem:



```
# single column analysis on price - summary of statistics
df_visualization['price_x'].describe()


count    22552.000000
mean        67.143668
std        220.266210
min          0.000000
25%         30.000000
50%         45.000000
75%         70.000000
max       9000.000000
Name: price_x, dtype: float64
```



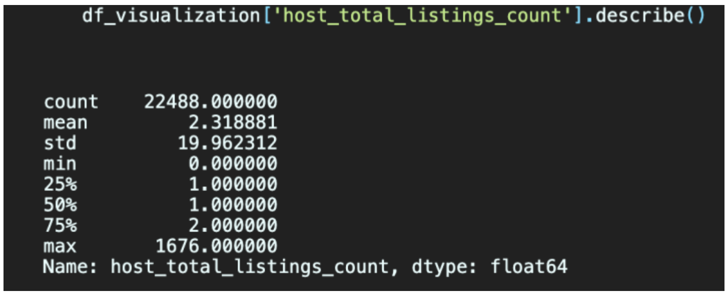
Then deciding to drop these rows that have more than 600 dollars of price.



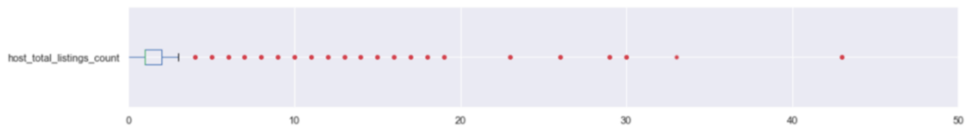
```
df_visualization.drop(df_visualization[ (df_visualization.price_x > 600) | (df_visualization.price_x == 0) ].index, axis=0, inplace=True)
```

Since the duplicate rows are already dropped with Openrefine here we had again 22552 rows in total that with dropping the outlier rows reduced to 22488.

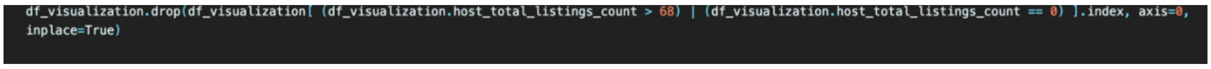
Again, finding new outliers in host_total_listings_count column:



```
df_visualization['host_total_listings_count'].describe()


count    22488.000000
mean         2.318881
std         19.962312
min          0.000000
25%          1.000000
50%          1.000000
75%          2.000000
max       1676.000000
Name: host_total_listings_count, dtype: float64
```



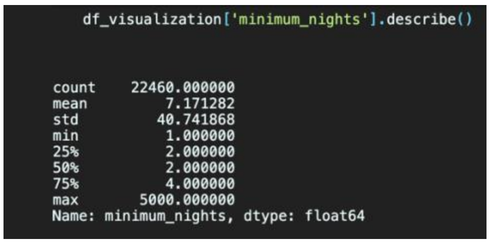
Dropping rows with more than 68 listings.



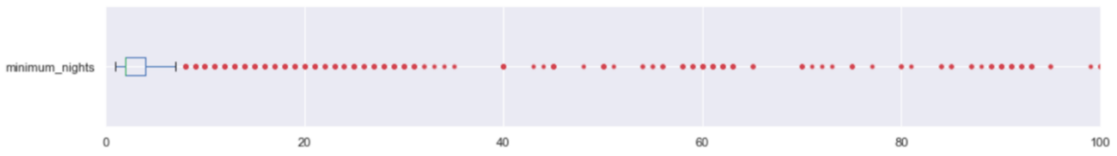
```
df_visualization.drop(df_visualization[ (df_visualization.host_total_listings_count > 68) | (df_visualization.host_total_listings_count == 0) ].index, axis=0,
inplace=True)
```

With dropping these outliers' number of rows reduced to 22460 rows.

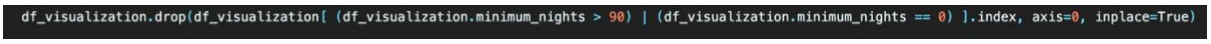
And lastly, finding outliers in minimum_nights column:



```
df_visualization['minimum_nights'].describe()


count    22460.000000
mean         7.171282
std         40.741868
min          1.000000
25%          2.000000
50%          2.000000
75%          4.000000
max       5000.000000
Name: minimum_nights, dtype: float64
```

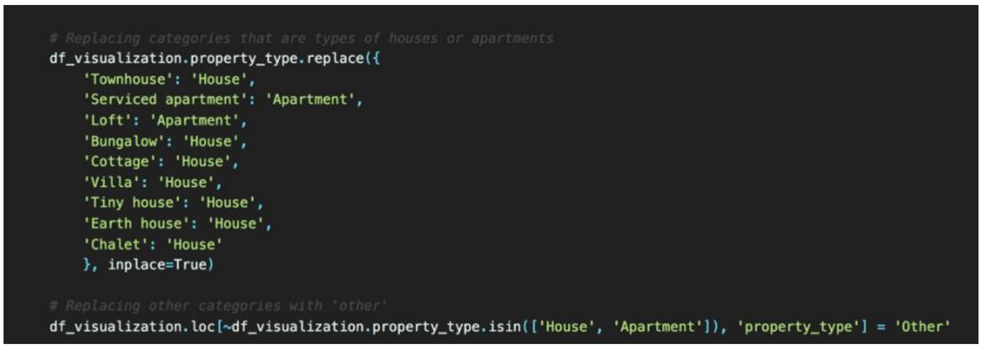


Dropping rows with more than 90 days.



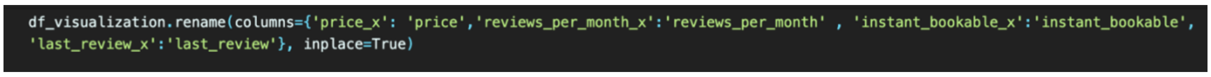
```
df_visualization.drop(df_visualization[ (df_visualization.minimum_nights > 90) | (df_visualization.minimum_nights == 0) ].index, axis=0, inplace=True)
```

With dropping the outliers in minimun_nights, number of rows reduced to 22312 rows. All other dimension problem which was made in goal B is solved and explained previously in OpenRefine, now we can continue with visualization. For this purpose, I changed the column property_type with only three values as below:



```
# Replacing categories that are types of houses or apartments
df_visualization.property_type.replace({
    'Townhouse': 'House',
    'Serviced apartment': 'Apartment',
    'Loft': 'Apartment',
    'Bungalow': 'House',
    'Cottage': 'House',
    'Villa': 'House',
    'Tiny house': 'House',
    'Earth house': 'House',
    'Chalet': 'House'
    }, inplace=True)

# Replacing other categories with 'other'
df_visualization.loc[~df_visualization.property_type.isin(['House', 'Apartment']), 'property_type'] = 'Other'
```

And renamed some of the columns to more accurate ones:



```
df_visualization.rename(columns={'price_x': 'price','reviews_per_month_x':'reviews_per_month' , 'instant_bookable_x':'instant_bookable',
'last_review_x':'last_review'}, inplace=True)
```

The visualization section is described in the presentation and its code is provided in the Jupyter Notebook. Both cleaned datasets whether by pandas or OpenRefine are available in the zip file as:

- Final_Cleaned_With_Pandas_GoalA.csv
- Final_Cleaned_With_openRefine_GoalA.csv + Recipe

In the end, All the described codes in this document for (Goal B, Goal A and Visualization) are available under the name of:

- Code_GoalB_GoalA_Visualizing.ipynb. **_Finish_**