Introduction

Python is an easy-to-learn programming language, which makes it the most preferred choice for beginners in Data Science, Data Analytics, and Machine Learning. It also has a great community of online learners and excellent data-centric libraries.

With so much data being generated, it becomes important that the data we use for Data Science applications like Machine Learning and Predictive Modeling is clean. But what do we mean by clean data? And what makes data dirty in the first place?

Dirty data simply means data that is erroneous. Duplicacy of records, incomplete or outdated data, and improper parsing can make data dirty. This data needs to be cleaned. Data cleaning (or data cleansing) refers to the process of “cleaning” this dirty data, by identifying errors in the data and then rectifying them.

Data cleaning is an important step in and Machine Learning project, and we will cover some basic data cleaning techniques (in Python) in this article.

Cleaning Data in Python

We will learn more about data cleaning in Python with the help of a sample dataset. We will use the Russian housing dataset on Kaggle.

https://www.kaggle.com/priyancagonsalves/account?verifyPhone=1&verifyPhoneReturnUrl=https%3A%2F%2Fwww.kaggle.com%2Fc%2Fsberbank-russian-housing-market%2Fdata

We will start by importing the required libraries.

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Download the data, and then read it into a Pandas DataFrame by using the read_csv() function, and specifying the file path.

Then use the shape attribute to check the number of rows and columns in the dataset. The code for this is as below:

```
df = pd.read_csv(train.csv')
df.shape
```

The dataset has 30,471 rows and 292 columns.

We will now separate the numeric columns from the categorical columns.

```
# select numerical columns
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
# select non-numeric columns
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
```

We are now through with the preliminary steps. We can now move on to data cleaning. We will start by identifying columns that contain missing values and try to fix them.

Missing values

We will start by calculating the percentage of values missing in each column, and then storing this information in a DataFrame.

```
# % of values missing in each column
values_list = list()
cols_list = list()
```

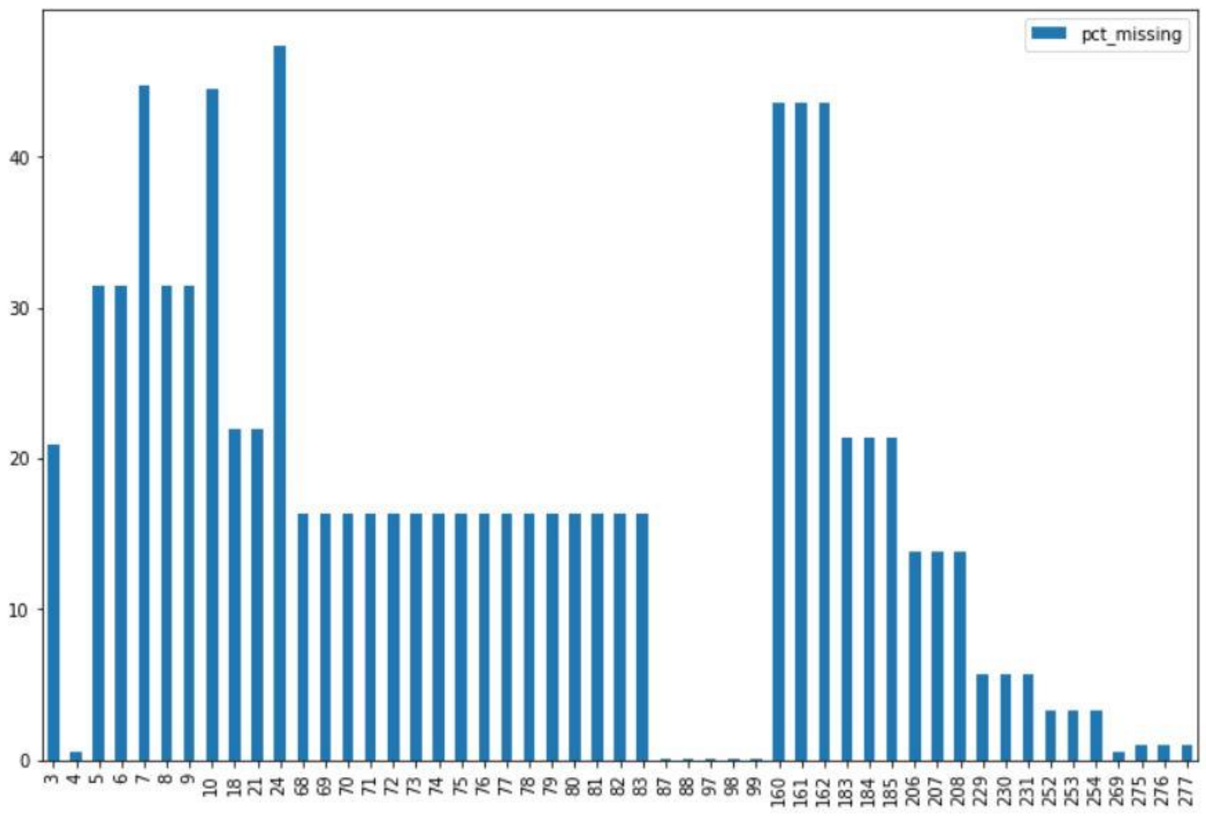
```
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())*100
    cols_list.append(col)
    values_list.append(pct_missing)
pct_missing_df = pd.DataFrame()
pct_missing_df['col'] = cols_list
pct_missing_df['pct_missing'] = values_list
```

The DataFrame pct_missing_df now contains the percentage of missing values in each column along with the column names.

We can also create a visual out of this information for better understanding using the code below:

```
pct_missing_df.loc[pct_missing_df.pct_missing > 0].plot(kind='bar',
figsize=(12,8))
plt.show()
```

The output after execution of the above line of code should look like this:



It is clear that some columns have very few values missing, while other columns have a substantial % of values missing. We will now fix these missing values.

There are a number of ways in which we can fix these missing values. Some of them are"

Drop observations

One way could be to drop those observations that contain any null value in them for any of the columns. This will work when the percentage of missing values in each column is very less. We will drop observations that contain null in those columns that have less than 0.5% nulls. These columns would be metro_min_walk, metro_km_walk, railroad_station_walk_km, railroad_station_walk_min, and ID_railroad_station_walk.

```
less_missing_values_cols_list =
list(pct_missing_df.loc[(pct_missing_df.pct_missing < 0.5) &
(pct_missing_df.pct_missing > 0), 'col'].values)
```

```
df.dropna(subset=less_missing_values_cols_list, inplace=True)
```

This will reduce the number of records in our dataset to 30,446 records.

Remove columns (features)

Another way to tackle missing values in a dataset would be to drop those columns or features that have a significant percentage of values missing. Such columns don't contain a lot of information and can be dropped altogether from the dataset. In our case, let us drop all those columns that have more than 40% values missing in them. These columns would be build_year, state, hospital_beds_raion, cafe_sum_500_min_price_avg, cafe_sum_500_max_price_avg, and cafe_avg_price_500.

```
# dropping columns with more than 40% null values

_40_pct_missing_cols_list =
list(pct_missing_df.loc[pct_missing_df.pct_missing > 40,
'col'].values)

df.drop(columns=_40_pct_missing_cols_list, inplace=True)
```

The number of features in our dataset is now 286.

Impute missing values

There is still missing data left in our dataset. We will now impute the missing values in each numerical column with the median value of that column.

```
df_numeric = df.select_dtypes(include=[np.number])

numeric_cols = df_numeric.columns.values

for col in numeric_cols:

    missing = df[col].isnull()

    num_missing = np.sum(missing)

    if num_missing > 0:  # impute values only for columns that have
missing values

        med = df[col].median() #impute with the median
        df[col] = df[col].fillna(med)
```

Missing values in numerical columns are now fixed. In the case of categorical columns, we will replace missing values with the mode values of that column.

```
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
for col in non_numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:  # impute values only for columns that have missing values
        mod = df[col].describe()['top'] # impute with the most frequently occuring value
        df[col] = df[col].fillna(mod)
```

All missing values in our dataset have now been treated. We can verify this by running the following piece of code:

```
df.isnull().sum().sum()
```

If the output is zero, it means that there are no missing values left in our dataset now.

We can also replace missing values with a particular value (like -9999 or ‘missing’) which will indicate the fact that the data was missing in this place. This can be a substitute for missing value imputation.

Outliers

An outlier is an unusual observation that lies away from the majority of the data. Outliers can affect the performance of a Machine Learning model significantly. Hence, it becomes important to identify outliers and treat them.

Let us take the ‘life_sq’ column as an example. We will first use the describe() method to look at the descriptive statistics and see if we can gather any information from it.

```
df.life_sq.describe()
```

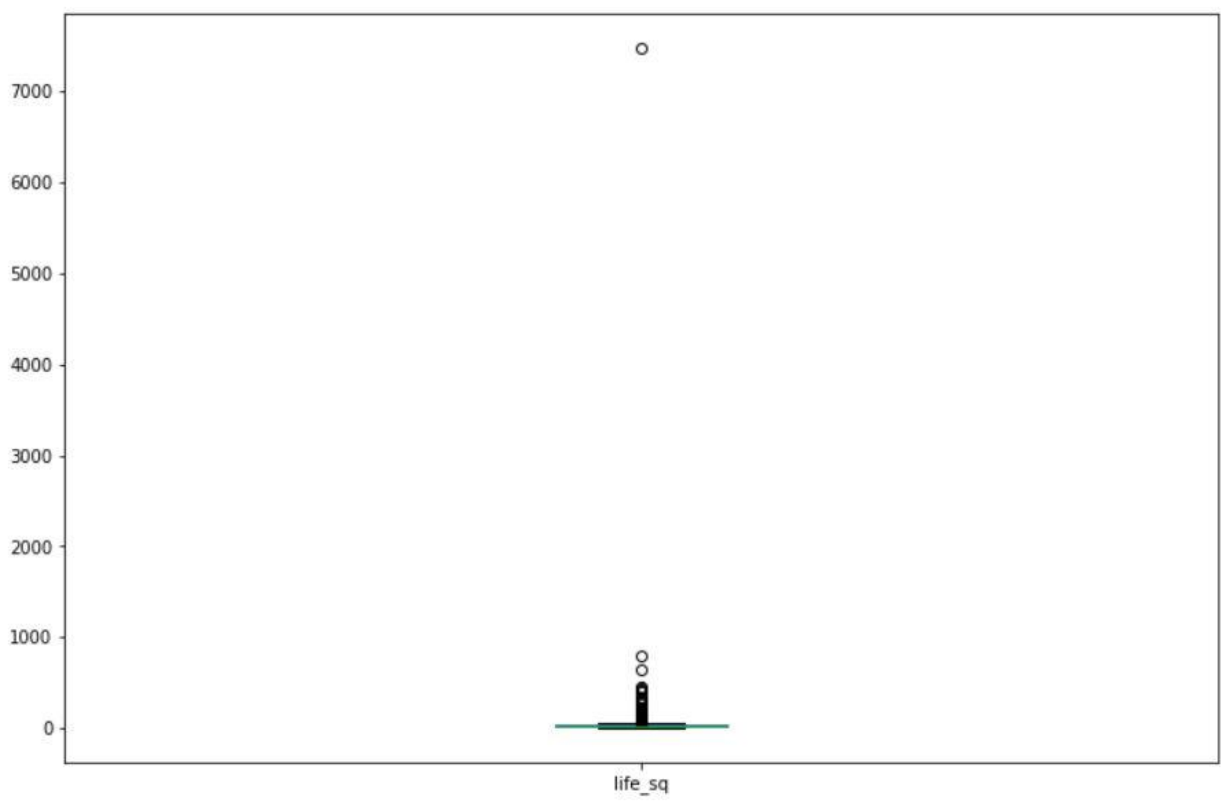
The output will look like this:

```
count    30446.000000
mean        33.482658
std         46.538609
min          0.000000
25%         22.000000
50%         30.000000
75%         38.000000
max       7478.000000
Name: life_sq, dtype: float64
```

From the output, it is clear that something is not correct. The max value seems to be abnormally large compared to the mean and median values. Let us make a boxplot of this data to get a better idea.

```
df.life_sq.plot(kind='box', figsize=(12, 8))
plt.show()
```

The output will look like this:



It is clear from the boxplot that the observation corresponding to the maximum value (7478) is an outlier in this data. Descriptive statistics, boxplots, and scatter plots help us in identifying outliers in the data.

We can deal with outliers just like we dealt with missing values. We can either drop the observations that we think are outliers, or we can replace the outliers with suitable values, or we can perform some sort of transformation on the data (like log or exponential). In our case, let us drop the record where the value of 'life_sq' is 7478.

```
# removing the outlier value in life_sq column
df = df.loc[df.life_sq < 7478]
```

Duplicate records

Data can sometimes contain duplicate values. It is important to remove duplicate records from your dataset before you proceed with any Machine

Learning project. In our data, since the ID column is a unique identifier, we will drop duplicate records by considering all but the ID column.

```
# dropping duplicates by considering all columns other than ID
cols_other_than_id = list(df.columns)[1:]
df.drop_duplicates(subset=cols_other_than_id, inplace=True)
```

This will help us in dropping the duplicate records. By using the shape method, you can check that duplicate records have actually been dropped. The number of observations is 30,434 now.

Fixing data type

Often in the dataset, values are not stored in the correct data type. This can create a problem in later stages, and we may not get the desired output or may get errors while execution. One common data type error is with dates. Dates are often parsed as objects in Python. There is a separate data type for dates in Pandas, called DateTime.

We will first check the data type of the timestamp column in our data.

```
df.timestamp.dtype
```

This returns the data type 'object'. We now know the timestamp is not stored correctly. To fix this, let's convert the timestamp column to the DateTime format.

```
# converting timestamp to datetime format
df['timestamp'] = pd.to_datetime(df.timestamp, format='%Y-%m-%d')
```

We now have the timestamp in the correct format. Similarly, there can be columns where integers are stored as objects. Identifying such features and correcting the data type is important before you proceed on to Machine Learning. Fortunately for us, we don't have any such issue in our dataset.

What we learned

In this article, we discussed some basic ways in which we can clean data in Python before starting with our Machine Learning project.

- We need to identify and remove missing values,
- identify and treat outliers,
- remove duplicate records,
- and fix the data type of all columns in our dataset

before we proceed with our ML task.