# Solution Review: Cleaning NYC Property Sales

This lesson provides the solutions to the data cleaning exercise in the previous lesson.

WE'LL COVER THE FOLLOWING

## 1. Change values

In this task we had to change the values in the `BOROUGH` column according to the following rule:

1 --> `Manhattan`

2 --> `Bronx`

3 --> `Brooklyn`

4 --> `Queens`

5 --> `Staten Island`

```
import pandas as pd
df = pd.read_csv('nyc_property_sales.csv')
# 1 --> Manhattan
condition = df['BOROUGH'] == 1
df.loc[condition,'BOROUGH'] = 'Manhattan'
# 2 --> Bronx
condition = df['BOROUGH'] == 2
df.loc[condition,'BOROUGH'] = 'Bronx'
# 3 --> Brooklyn
condition = df['BOROUGH'] == 3
df.loc[condition,'BOROUGH'] = 'Brooklyn'
# 4 --> Queens
condition = df['BOROUGH'] == 4
df.loc[condition,'BOROUGH'] = 'Queens'
df.loc[condition,'BOROUGH'] = 'Queens'
# 5 --> Staten Island
condition = df['BOROUGH'] == 5
df.loc[condition,'BOROUGH'] = 'Staten Island'

print(df['BOROUGH'].unique())
```

By looking at the problem statement, we can see that we need to write similar code for all 5 categories. We do each category one by one.

To change all instances of a specific value in a column, first, we need to find the rows where that value is present. To do this, we write our condition in **line 4**. `df['BOROUGH'] == 1`, gives us a list of `True`/`False` against each row. It is true for rows where the value of the `BOROUGH` column is 1. Now we need to go to all these places and change the value. We do that in **line 5** by using `loc`. We index the dataframe by the rows we had stored in `condition` and by the column `BOROUGH`.

`df.loc[condition,'BOROUGH']` gives us all the cells where the value for `BOROUGH` is 1. So, we set these values to `Manhattan`. We follow the same steps for all other values.

In the end, we can verify our results by the output of **line 19**. It gives us the unique values that the column `BOROUGH` takes.

## 2. Missing Values

In this task, we had to remove rows that had missing values in `SALE PRICE` column.

```
import pandas as pd
df = pd.read_csv('nyc_property_sales.csv')

condition = df['BOROUGH'] == 1
df.loc[condition,'BOROUGH'] = 'Manhattan'

condition = df['BOROUGH'] == 2
df.loc[condition,'BOROUGH'] = 'Bronx'

condition = df['BOROUGH'] == 3
df.loc[condition,'BOROUGH'] = 'Brooklyn'

condition = df['BOROUGH'] == 4
df.loc[condition,'BOROUGH'] = 'Queens'
```

```
condition = df['BOROUGH'] == 5
df.loc[condition,'BOROUGH'] = 'Staten Island'

print(df.shape)

# Solution
present = df['SALE PRICE'].notnull()
df = df[present]

print(df.shape)
```

This task is simple. To remove the rows containing missing values in the `SALE PRICE`, we just find the rows that do not contain missing values in `SALE PRICE` by using the `notnull` function in **line 22**. It gives us a list of `True`/`False` against each row. It is true for rows where there is no missing value. We store this list in `present`. In the next line, we use this list to filter the rows that have missing values. We verify the results by looking at the dimensions of the dataframe before (**line 19**) and after (**line 25**) filtering.

## 3. Duplicate Values #

In this task, we had to remove duplicate rows.

```
import pandas as pd
df = pd.read_csv('nyc_property_sales.csv')

condition = df['BOROUGH'] == 1
df.loc[condition,'BOROUGH'] = 'Manhattan'

condition = df['BOROUGH'] == 2
df.loc[condition,'BOROUGH'] = 'Bronx'

condition = df['BOROUGH'] == 3
df.loc[condition,'BOROUGH'] = 'Brooklyn'

condition = df['BOROUGH'] == 4
df.loc[condition,'BOROUGH'] = 'Queens'

condition = df['BOROUGH'] == 5
df.loc[condition,'BOROUGH'] = 'Staten Island'

present = df['SALE PRICE'].notnull()
df = df[present]

print(df.shape)

# Solution
df = df.drop_duplicates(subset=df.columns)
```

```
print(df.shape)
```

Recall that we use the function `drop_duplicates` to remove duplicates. We have to provide it a subset of columns for which the function checks if the values are duplicated in all these subset columns. If they are, it removes the duplicates. In our case, we used `drop_duplicates` in **line 25**. We provide our whole list of columns, which we access by `df.columns`, to the function. We verify the results by looking at the dimensions of the dataframe before (**line 22**) and after (**line 27**) removing duplicate rows.

## 4. Outliers

In this task, we had to remove outliers using the Interquartile range, but there was a catch. The quantiles for q1 and q3 were 0.10 and 0.90. The columns in which we had to check for outliers were:

- `RESIDENTIAL UNITS`
- `COMMERCIAL UNITS`
- `TOTAL UNITS`
- `LAND SQUARE FEET`
- `GROSS SQUARE FEET`
- `YEAR BUILT`

```
import pandas as pd
df = pd.read_csv('nyc_property_sales.csv')

condition = df['BOROUGH'] == 1
df.loc[condition,'BOROUGH'] = 'Manhattan'

condition = df['BOROUGH'] == 2
df.loc[condition,'BOROUGH'] = 'Bronx'

condition = df['BOROUGH'] == 3
df.loc[condition,'BOROUGH'] = 'Brooklyn'

condition = df['BOROUGH'] == 4
df.loc[condition,'BOROUGH'] = 'Queens'

condition = df['BOROUGH'] == 5
df.loc[condition,'BOROUGH'] = 'Staten Island'

present = df['SALE PRICE'].notnull()
```

```
df = df[present]

df = df.drop_duplicates(subset=df.columns)


print(df.shape)

# Solution
## Retrieve only outlier columns
new_df = df[['RESIDENTIAL UNITS', 'COMMERCIAL UNITS','TOTAL UNITS', 'LAND SQUARE FEET','GROSS

## find max and min using IQR
Q1 = new_df.quantile(0.10)
Q3 = new_df.quantile(0.90)
IQR = Q3-Q1
minimum = Q1 - 1.5*IQR
maximum = Q3 + 1.5*IQR

## condition on which to filter
condition = (new_df <= maximum) & (new_df >= minimum)
condition = condition.all(axis=1)

## Filter rows that have outliers
df = df[condition]

print(df.shape)
```

In this task, we need to

- Find the minimum and maximum boundary values within which values are allowed for each column.
- Find rows that satisfy the boundary values condition for all columns.
- Filter for rows that are not outliers.

First, we create a new dataframe `new_df` with only columns for which we have to check for outliers in **line 28**. Step 1 is done in **lines 31-35**. We use the `quantile` function to get quantiles and find maximum and minimum boundary values.

Step 2 starts on **line 38**. It gives us a dataframe that has `True` or `False` for every cell in `new_df` based on if it satisfies the condition or not. In the **next line**, we specify that the condition should be true for all three columns by using the `all` function with `axis=1` argument. This gives us a list of `True`/`False` against each row. If a row has all `True` values, then it gives a `True` value to that row. Here, we have a list of rows with which we can filter. We filter the original dataframe `df` on **line 42**. We verify the results by looking at the dimensions of the dataframe before (**line 34**) and after (**line 44**) removing the dimensions of the dataframe before **(line 24)** and after **(line 44)** removing duplicate rows.

Before cleaning this dataset, we had 84548 rows in the data. The number reduced to 39902 after cleaning. This just shows how much redundant data we had in the data set.

This was it from this chapter. In the next chapter, we will cover *Exploratory Data Analysis*.