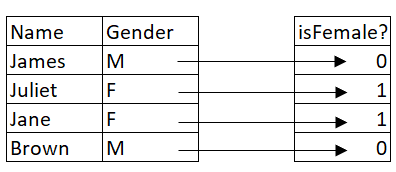
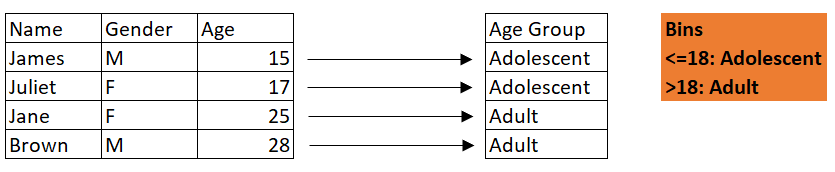
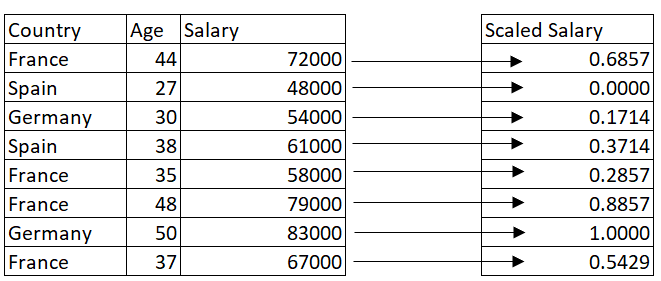
* **Encoding:** This is a process of converting categorical values into binary values which can then be used in computations for machine learning. The figure shows what the gender column would look like if it were encoded. While it still carries the same information, it has been transformed into a different format.



* **Data binning**: Also referred to as bucketing, this process helps to reduce the effect/size of minor observations. It entails grouping continuous values into bins (or categories).



* **Scaling**: This is a common technique in machine learning to help standardize values of different scales into a fixed range.



we have a dataset provided for us [**https://www.nyc.gov/site/finance/taxes/property-rolling-sales-data**](https://www.nyc.gov/site/finance/taxes/property-rolling-sales-data)**.** page. The Department of Finance's Rolling Sales files list properties that sold in the last twelve-month period in **New York City** for tax classes 1, 2, and 4.

#pandas

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

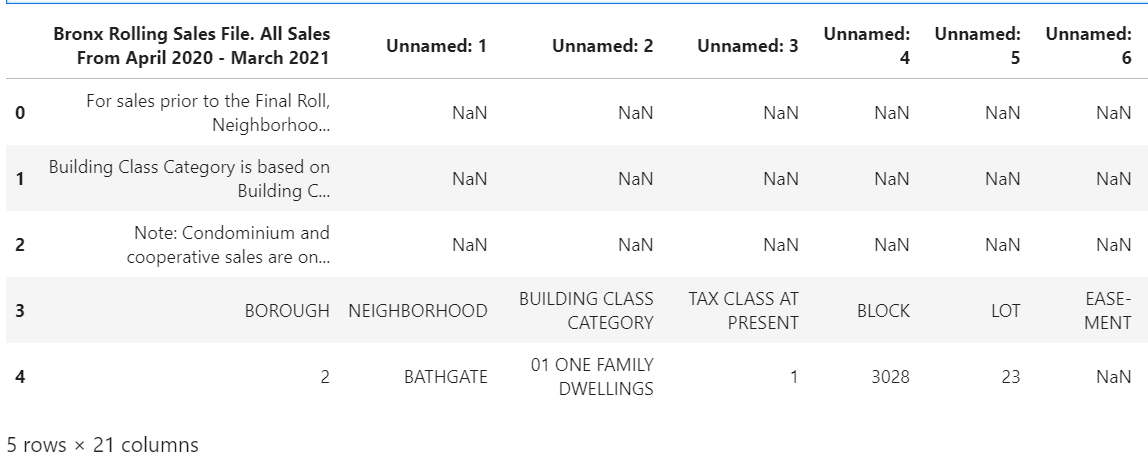
#read excel file

df = pd.read\_excel("rollingsales\_bronx.xls")

#preview data

df.head()

OpenAI



The notable issues with this data frame include:

1. unnamed columns
2. missing values in the first three rows

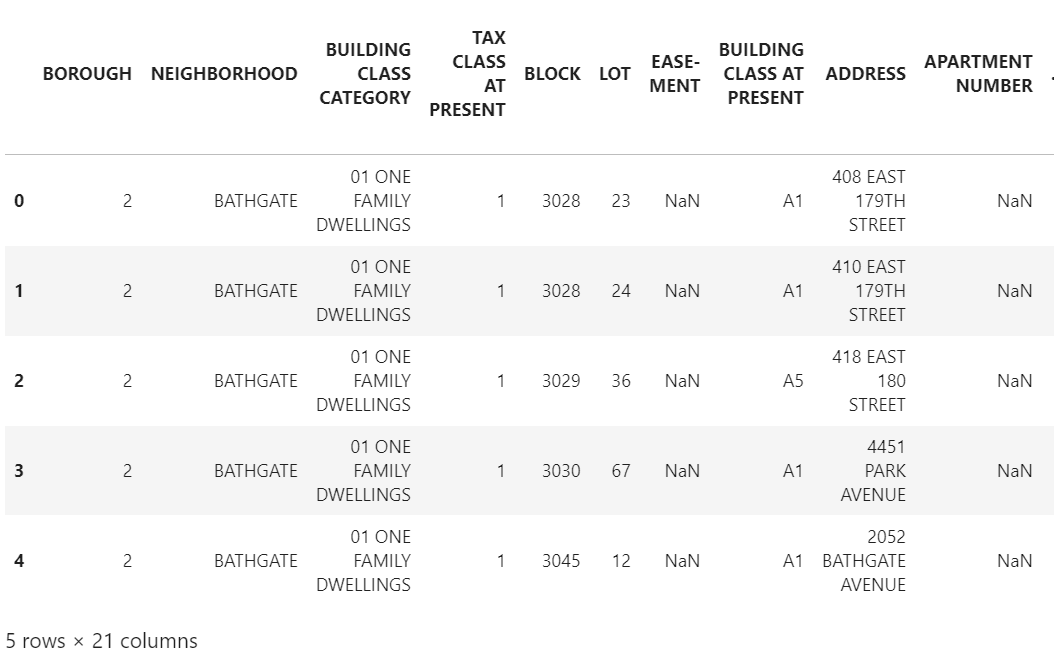
However, exploring the actual data in excel reveals that the actual table in the workbook doesn't start until the 4th row. Therefore, reading the data again but setting a few parameters allows us to indicate which rows to skip and which row to make into the header:

#we read the data in again but skill 3 rows this time

df = pd.read\_excel("rollingsales\_bronx.xls", skiprows = 4, header = [0])

df.head()

OpenAI



There is a significant change in how data is now displayed. While we could have cleaned the first import, data exploration has helped us save ample time with just a few modifications to our data import function in Python.

*Data exploration is like walking into a crime scene as an investigative agent, where we passively observe all things out of place and data cleaning is the active process of solving the actual crime.*

**Data Cleaning**

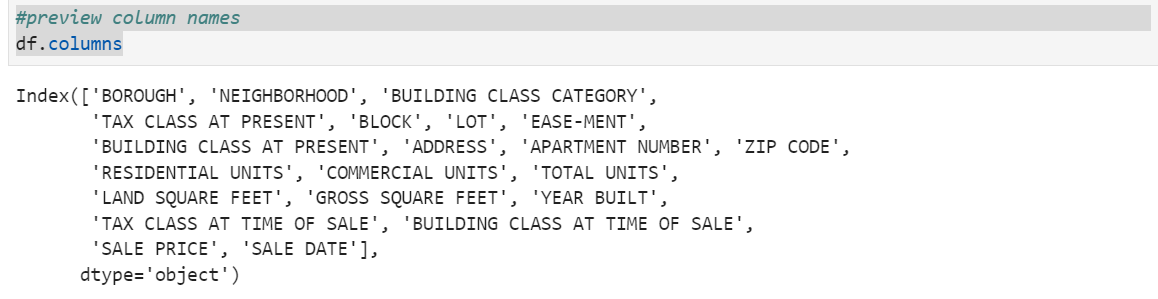
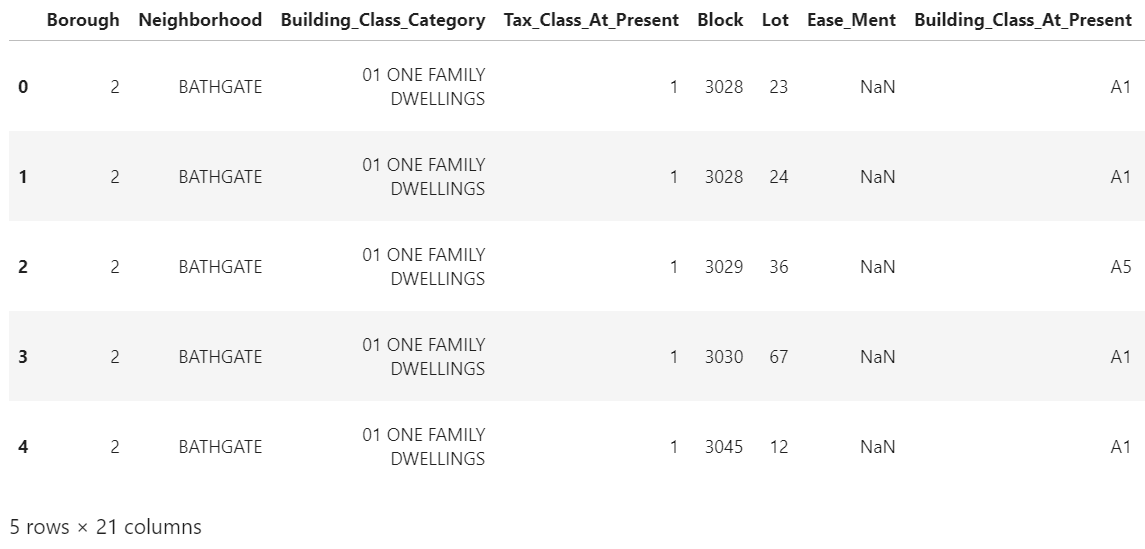
Data exploration will typically go hand in hand with data cleaning processes. Once we uncover issues with our data using visual, aggregation, or statistical means, we need to fix them. In this phase, data will either be removed, corrected, or imputed.

**Inconsistent Records**

A typical example of inconsistent records is a form where individuals are required to enter their gender. We can have different formats representing the same thing e.g 'M', 'm', 'Male', or 'MALE', and did we sell 'aples', 'apples', or 'APPLES' this month. The best way to spot them can be via a frequency chart or making a distinct display of all values in the column. Some operations on inconsistent records will include but are not limited to:

* Converting strings to lower or proper case
* Removing white spaces
* Renaming column names

**Python example:** Using the dataset imported earlier under data exploration, notice how the column names are inconsistently written:

While this doesn't seem to be a big deal, it is important to make column names consistently accessible. For example, **EASE-MENT** has a hyphen (-) between each word, while the rest just have spaces. Part of data cleaning is ensuring consistency. Let us attempt to fix this in two steps: 1. Replace spaces between words with an underscore 2. Convert to proper case #Extract columns cols = df.columns #Create empty list new\_cols = [] #iterate to fix issues with names for column in cols: #to proper case proper\_cols = column.title() #replace space/hyphen with underscore proper\_cols\_hyphen = proper\_cols.replace(" ", "\_") clean\_col = proper\_cols\_hyphen.replace("-", "\_") #append to empty list new\_cols.append(clean\_col) #diplay columns new\_cols #replace existing columns in dataframe with new df.columns = new\_cols #preview df.head()

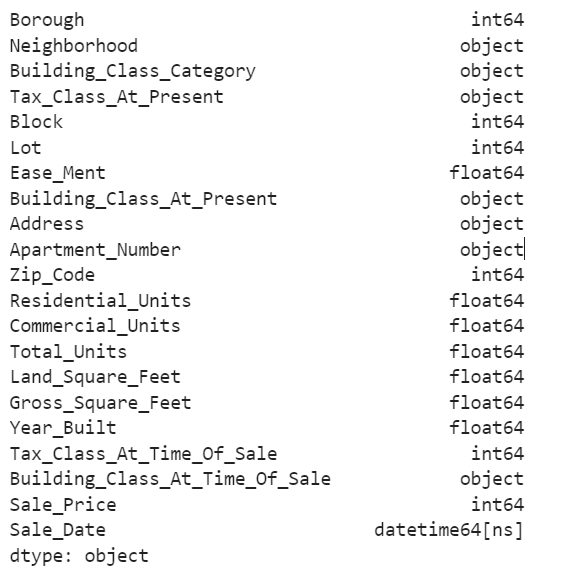
**Data Types and Type Conversion**

The data type is simply how data is represented, which tells the compiler how the data is to be used. How data is represented determines the kind of operations possible with such data. Let us preview the data types in each column of our dataframe from the Bronx data.

#data types

df.dtypes

OpenAI



**NB:** Consider how we are now able to easily pick column names using the dot method (df.Year\_Built).

All data types appear in order except the **Year\_Built** column, which should be categorical. While this has no major significance, when we aggregate these columns **Year\_Built** will likely be included.

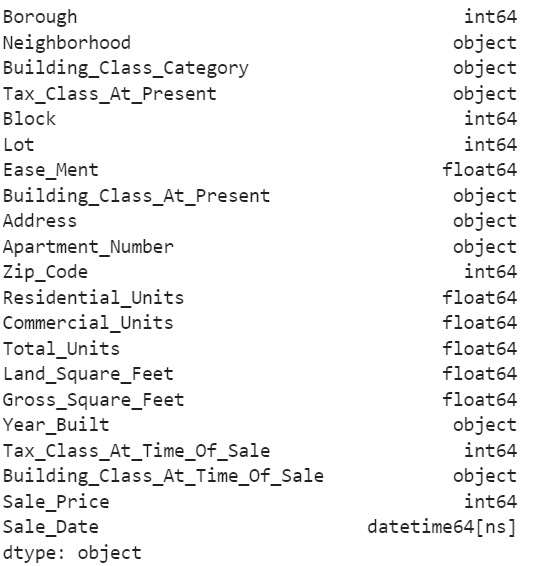
#Fix Year\_Built column

df.Year\_Built = df.Year\_Built.astype("str")

#Check

df.dtypes

OpenAI



**Missing Data**

There are times when there are observations (rows) with missing data and sometimes you can have entire columns with completely missing values. So what do we do in instances like this? Do we ignore it? This can be likened to a leaking roof and when it rains, it leaks.

Missing values have different representations across different tools. These representations include NULL, empty strings, NaN, NA, #NA, etc. Data context is important in dealing with missing data, and understanding why data is missing is crucial. There are two primary methods for dealing with missing data:

* Removal of data
* Imputation of data

Before any decision can be made on what method will be used, one must first examine the patterns of missing values across columns. We will use visual and descriptive techniques to assess these.

#extract the column names

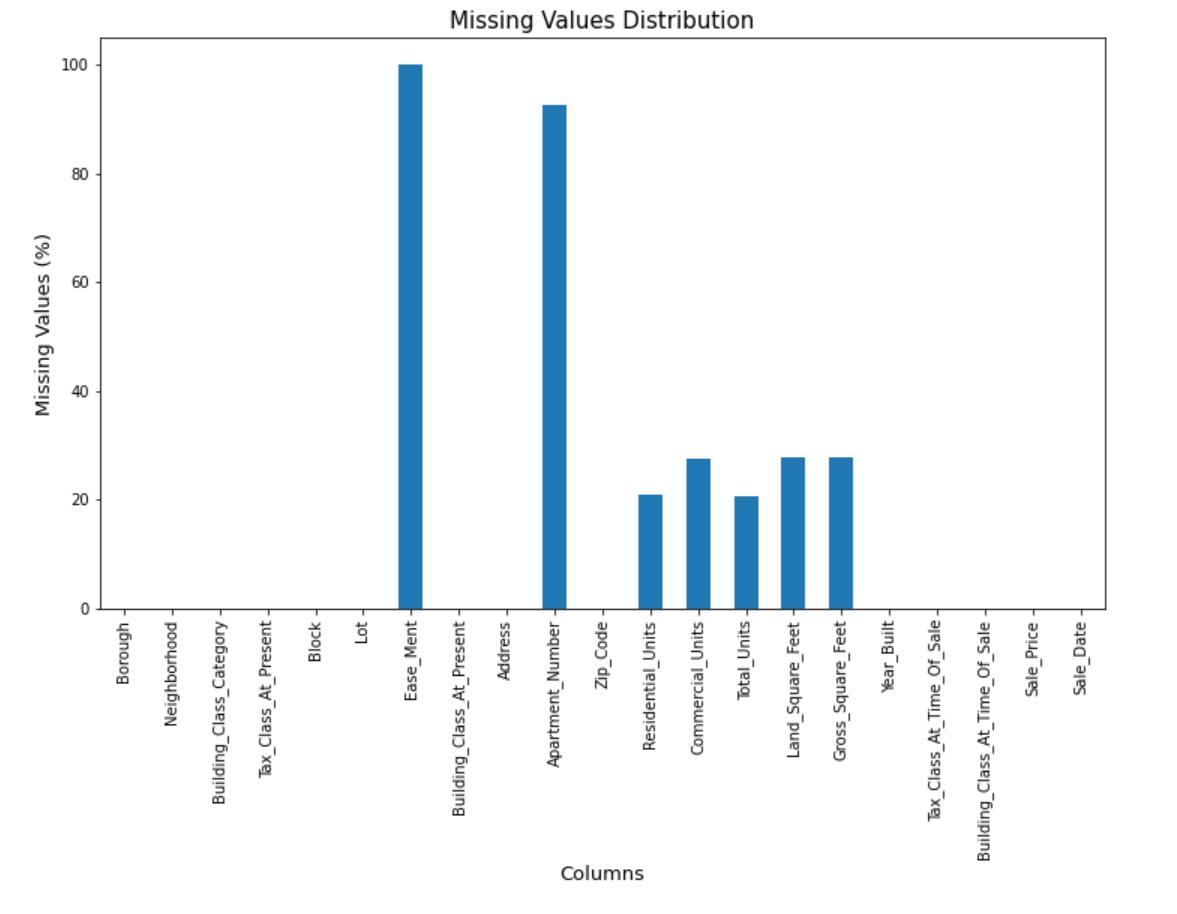
cols = df.columns

#plot a heatmap of missing values with seaborn

plt.figure(figsize = (10,5))

sns.heatmap(df[cols].isnull())

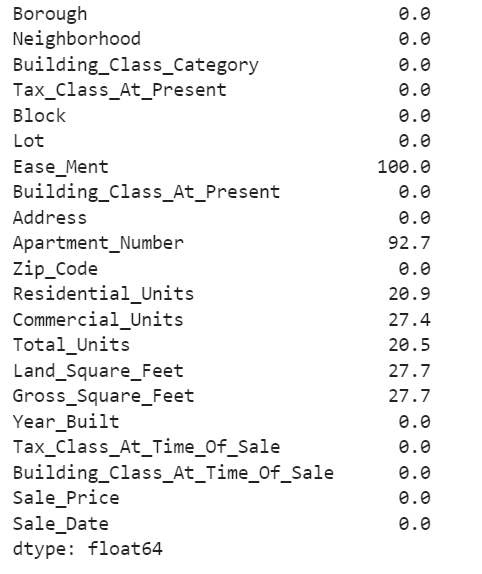
OpenAI



The plot visually shows that:

* All values in **Ease\_ment** and almost all values in **Apartment\_Number** are missing
* The **Units** and **Square\_Feet**-related columns have missing values spread out For much larger datasets, this plot might not be ideal. Another way to handle this is to examine how significant these missing values are compared to the actual data size.
* #Get the percentage of missing values in each column
* missing\_pct = round(df.isnull().sum()/len(df) \* 100, 1)
* print(missing\_pct)

OpenAI



We will drop columns Ease\_Ment and Apartment\_Number and then replace the missing values in the remaining columns.

#columns with more than 30% missing values

drop\_cols = missing\_pct[missing\_pct > 30].index

#We can drop these columns with greater than 30 percent missing values

df\_new = df.drop(columns = drop\_cols)

#Extract columns with mising values between 1 and 30%

replace\_cols = missing\_pct[(missing\_pct > 0) & (missing\_pct < 30)].index

#Iterate to replace missing values

for col in replace\_cols:

#if column is year built we replace with median otherwise the mean

if col == "Year\_Built":

df\_new.fillna(df\_new[col].median(), inplace = True)

else:

mean\_value = df\_new[col].mean()

df\_new.fillna(mean\_value, inplace = True)

df\_new.isnull().sum() #preview for missing values

OpenAI

