data-cleaning

April 11, 2024

1 Data Cleaning by Shromana Majumder 11/04/24

Data cleaning is the process of changing or eliminating garbage, incorrect, duplicate, corrupted, or incomplete data in a dataset. It is a crucial step in preparing your data for analysis, visualization, or further processing. It involves identifying and correcting errors, inconsistencies, and inaccuracies in datasets to ensure they are accurate and reliable.

1. **Why Data Cleaning Matters**:
    - **Quality Matters**: The quality of data significantly impacts the quality of insights and results.
    - **Python's Role**: Python provides a robust environment for data cleaning, thanks to libraries like **pandas** and **NumPy**.
    - **Automation**: While manual data cleaning using tools like Excel is possible, Python allows for automation, making it ideal for larger datasets and repetitive tasks.
2. **Python Libraries for Data Cleaning**:
    - **pandas**: A powerful library for data manipulation and analysis. It provides functions to handle missing values, duplicates, and more.
    - **NumPy**: A fundamental package for numerical computations. It's useful for handling arrays and matrices efficiently.
3. **Common Data Cleaning Tasks in Python**:
    - **Dropping Unnecessary Columns**: Remove columns that are not relevant to analysis.
    - **Changing Index**: Modify the index of a DataFrame.
    - **Cleaning Text Columns**: Use `.str()` methods to clean text data (e.g., removing whitespace, converting to lowercase).
    - **Applying Functions Element-Wise**: Use `DataFrame.applymap()` to clean the entire dataset element-wise.
    - **Renaming Columns**: Give more recognizable labels to columns.
    - **Skipping Rows in CSV Files**: Skip unnecessary rows when reading data from CSV files.

Clean data leads to better insights and more accurate results.

1.1 Data Cleaning Cycle

1. Import Data
2. Merge Dataset
3. Rebuild Missing Data
4. Standarize

5. Normalize
6. De Duplicate
7. Verify & Enrich
8. Export data

1.2 Understanding Dirty Data

- There are many types of errors and inconsistencies that can contribute to data being dirty.

1.3 Missing values

Missing values can have multiple effects on analysis. Large portions of crucial data missing can cause bias in results. Additionally, NaN values or missing cells in a DataFrame may break some Python code

1.4 Outliers

Outliers are values that are far outside the norm and not representative of the data. Outliers can skew results, ultimately suggesting the wrong answer.

1.5 Duplicates

Duplicate data entries can overrepresent one entry in analysis, leading to the wrong conclusion.

1.6 Erroneous data

Sometimes, the values in data set are simply wrong. We may have the wrong spelling for a customer's name, the wrong product number, outdated information, or incorrectly labeled data.

1.7 Inconsistencies

Inconsistency in the format of the data. Different values may be reported in different units (kilometers vs. miles vs. inches), in different styles (Month Day, Year vs. Day-Month-Year), in different data types (floats vs. integers), or even in different file types (.jpg vs. .png).

2 Understanding of the data

1. First and foremost, look at the source of dataset and determine if that source has any bias or agenda that may affect the quality or reliability of data.
2. Learn the context of data and any other factors that may have affected data that aren't internally accounted for.
3. Determine how many different variables we have. In a table-formatted dataset, the variables are typically the columns, while each data entry is a row.
4. Determine how many different categories within each variable you have.
5. Look at the summary statistics for each column, including the mean, median, variance, and standard deviation.

3 Techniques for Data Cleaning in Python

[2]:
```
import numpy as np
import pandas as pd
```

```
C:\Users\shromana\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60:
UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version
'1.3.5' currently installed).
  from pandas.core import (
```

[3]:
```
import os
import re
import missingno as msn
```

3.0.1 Load dataset

[18]:
```
df = pd.read_csv("IRIS.csv")
df.head()
```

[18]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

3.1 Handling missing values

3.1.1 Using isnull() function:

[43]:
```
df.isnull()
```

[43]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
..	…	…	…	…	…
145	False	False	False	False	False
146	False	False	False	False	False
147	False	False	False	False	False
148	False	False	False	False	False
149	False	False	False	False	False

[150 rows x 5 columns]

3.1.2 Using isna() function:

[44]:
```
df.isna()
```

[44]:
```
     sepal_length  sepal_width  petal_length  petal_width  species
0           False        False         False        False    False
1           False        False         False        False    False
2           False        False         False        False    False
3           False        False         False        False    False
4           False        False         False        False    False
..            …            …            …          …       …
145         False        False         False        False    False
146         False        False         False        False    False
147         False        False         False        False    False
148         False        False         False        False    False
149         False        False         False        False    False

[150 rows x 5 columns]
```

3.1.3 Using isna().any()

[45]:
```
df.isna().any()
```

[45]:
```
sepal_length    False
sepal_width     False
petal_length    False
petal_width     False
species         False
dtype: bool
```

3.1.4 Using isna(). sum()

[46]:
```
df.isna().sum()
```

[46]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

3.1.5 Using isna().any().sum()

[47]:
```
df.isna().any().sum()
```

[47]:
```
0
```

3.1.6 Identify rows with NaN values

[19]:
```
rows_with_nan = df[df.isnull().any(axis=1)]

#View the rows with NaN values
print(rows_with_nan)
```

```
Empty DataFrame
Columns: [sepal_length, sepal_width, petal_length, petal_width, species]
Index: []
```

3.1.7 Percent of data that is missing

[ ]:
```
missing_values_count = df.isnull().sum()

total_cells = np.column(df.shape)
total_missing = missing_values_count.sum()

# percent of data that is missing
percent_missing = (total_missing/total_cells) * 100
print(percent_missing)
```

3.2 Drop missing values

- Deletion is the easiest way to deal with entries that contain missing values.

[20]:
```
df.dropna(inplace=True)
```

3.2.1 Filling in missing values

- Preferred method of handling missing values is imputing a reasonable value.

The .fillna() method from Pandas will impute missing values.

3.2.2 Another approach

[24]:
```
df1 = df.drop(columns=['species'])
```

replace all NA's with 0

[48]:
```
df1.fillna(0)
```

[48]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
..	…	…	…	…

```
145            6.7          3.0           5.2          2.3
146            6.3          2.5           5.0          1.9
147            6.5          3.0           5.2          2.0
148            6.2          3.4           5.4          2.3
149            5.9          3.0           5.1          1.8

[150 rows x 4 columns]
```

[26]:
```
df1.fillna(df1.mean(), inplace=True)
```

[27]:
```
from sklearn.impute import SimpleImputer
```

[28]:
```
imputer = SimpleImputer(strategy='mean')
df_imputed = pd.DataFrame(imputer.fit_transform(df1), columns=df1.columns)
```

3.3 Outlier detection and treatment

- A common method is to calculate a Z-score for each data point and eliminate the values with an extreme Z-score.

[29]:
```
# Generate some sample data
np.random.seed(0)
data = np.random.randint(low=0, high=11, size=1000)

# Add some outliers
data[0] = 100
data[1] = -100

# Calculate Z-scores
z_scores = (data - np.mean(data)) / np.std(data)

# Identify outliers based on Z-score threshold (e.g., 3)
threshold = 3
outliers = np.where(np.abs(z_scores) > threshold)[0]

print("Outliers identified using Z-score method:")
print(data[outliers])
```

```
Outliers identified using Z-score method:
[ 100 -100]
```

- Another method is to calculate the interquartile range (IQR) of the distribution and classify any values that are Q1-(1.5 x IQR) or Q3 + (1.5 x IQR) as potential outliers.

[31]:
```
# Generate some sample data
np.random.seed(0)
data = np.random.randint(low=0, high=11, size=1000)
# Add some outliers
```

```
data[0] = 100
data[1] = -100

# Calculate quartiles and IQR
q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)
iqr = q3 - q1

# Identify outliers based on IQR
lower_bound = q1 - (1.5 * iqr)
upper_bound = q3 + (1.5 * iqr)
outliers = np.where((data < lower_bound) | (data > upper_bound))[0]

print("Outliers identified using IQR method:")
print(data[outliers])
```

```
Outliers identified using IQR method:
[ 100 -100]
```

- It may be possible to remove the outlier from the dataset or replace it with a less extreme value that retains the overall shape of the distribution.

Capping is a method where wenset a cap, or threshold, on data's distribution and replace any values outside those bounds with a specified value.

[32]:
```
# Create a sample DataFrame with outliers
data = {
    'A': [100, 90, 85, 88, 110, 115, 120, 130, 140],
    'B': [1, 2, 3, 4, 5, 6, 7, 8, 9]
}
df2 = pd.DataFrame(data)

# Define the lower and upper thresholds for capping (Here I used the 5th and␣
 ↪95th percentiles)
lower_threshold = df2.quantile(0.05)
upper_threshold = df2.quantile(0.95)

# Cap outliers
capped_df = df2.clip(lower=lower_threshold, upper=upper_threshold, axis=1)

print("Original DataFrame:")
print(df2)
print("\nCapped DataFrame:")
print(capped_df)
```

```
Original DataFrame:
     A  B
0  100  1
1   90  2
```

```
2   85   3
3   88   4
4  110   5
5  115   6
6  120   7
7  130   8
8  140   9

Capped DataFrame:
       A    B
0  100.0  1.4
1   90.0  2.0
2   86.2  3.0
3   88.0  4.0
4  110.0  5.0
5  115.0  6.0
6  120.0  7.0
7  130.0  8.0
8  136.0  8.6
```

3.4 Parsing date

[49]:
```python
# Creating a DataFrame
data = {'date': ['2024-04-12', '2024-04-13', '2024-04-14', '2024-04-15']}
df4 = pd.DataFrame(data)
```

[50]:
```python
# check the data type  date column
df4['date'].dtype
```

[50]:
```
dtype('O')
```

3.4.1 Convert date columns to datetime

[53]:
```python
df4['date_parsed'] = pd.to_datetime(df4['date'], format="%Y-%m-%d")
```

[54]:
```python
print(df4)
```

```
         date date_parsed
0  2024-04-12  2024-04-12
1  2024-04-13  2024-04-13
2  2024-04-14  2024-04-14
3  2024-04-15  2024-04-15
```

3.5 Character Encodings

- Character encodings are specific sets of rules for mapping from raw binary byte strings to characters that make up human-readable text .

- UTF-8 is the standard text encoding.

3.6 Addressing inconsistencies

1. Unit conversion
2. Email, phone, and address standardization
3. Removing punctuation from strings
4. Using value mapping to address common abbreviations

3.7 Inconsistent Data

[56]:
```python
# Example DataFrame with inconsistent data
data = {'gender': ['male', 'Female', 'Male', 'female']}
df5 = pd.DataFrame(data)

# Convert to lowercase
df5['gender'] = df5['gender'].str.lower()
print(df5)
```

```
   gender
0    male
1  female
2    male
3  female
```

3.8 Convert data types

[57]:
```python
data = {'sales': ['100', '200', '300']}
df6 = pd.DataFrame(data)
# Convert 'sales' to numeric
df6['sales'] = pd.to_numeric(df6['sales'])
```

3.9 Dealing with Duplicates

- Using the drop_duplicates() method in Pandas

[33]:
```python
duplicate_rows = df1[df1.duplicated()]
```

[34]:
```python
cleaned_df = df.drop_duplicates()
```

- In some cases, it might be more appropriate to merge duplicate records

[39]:
```python
# Sample DataFrame
data = {
    'customer_id': [102, 102, 101, 103, 102],
    'product_id': ['A', 'B', 'A', 'C', 'B'],
    'quantity_sold': [5, 3, 2, 1, 4]
}
```

```
df3 = pd.DataFrame(data)
df3
```

[39]:

```
   customer_id product_id  quantity_sold
0          102          A              5
1          102          B              3
2          101          A              2
3          103          C              1
4          102          B              4
```

[42]:

```
# Merging duplicates by aggregating values
merged_df = df3.groupby(['customer_id', 'product_id']).agg({'quantity_sold':␣
 ↪'sum'}).reset_index()

merged_df
```

[42]:

```
   customer_id product_id  quantity_sold
0          101          A              2
1          102          A              5
2          102          B              7
3          103          C              1
```

Things to remember before transforming data.

1. Understand the underlying data distribution
2. Choose an appropriate transformation
3. Handle zeros and negative values
4. Validate transformed data

3.10 Loading Another Dataset

[4]:

```
df = pd.read_csv("my_file (1).csv")
df.head()
```

[4]:

```
   Rank  Peak All Time Peak  Actual gross Adjusted gross (in 2022 dollars)  \
0     1     1             2  $780,000,000                     $780,000,000
1     2     1          7[2]  $579,800,000                     $579,800,000
2     3  1[4]          2[5]  $411,000,000                     $560,622,615
3     4  2[7]         10[7]  $397,300,000                     $454,751,555
4     5  2[4]           NaN  $345,675,146                     $402,844,849

         Artist                  Tour title    Year(s)  Shows  Average gross  \
0  Taylor Swift              The Eras Tour †  2023-2024     56    $13,928,571
1       Beyoncé        Renaissance World Tour       2023     56    $10,353,571
2       Madonna  Sticky & Sweet Tour ‡[4][a]  2008-2009     85     $4,835,294
3          Pink  Beautiful Trauma World Tour  2018-2019    156     $2,546,795
4  Taylor Swift       Reputation Stadium Tour       2018     53     $6,522,173
```

```
  Ref.
0  [1]
1  [3]
2  [6]
3  [7]
4  [8]
```

[5]:
```python
df.describe()
```

[5]:

	Rank	Shows
count	20.000000	20.000000
mean	10.450000	110.000000
std	5.942488	66.507617
min	1.000000	41.000000
25%	5.750000	59.000000
50%	10.500000	87.000000
75%	15.250000	134.500000
max	20.000000	325.000000

[6]:
```python
df.shape
```

[6]: (20, 11)

[7]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 11 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Rank                               20 non-null     int64
 1   Peak                               9 non-null      object
 2   All Time Peak                      6 non-null      object
 3   Actual gross                       20 non-null     object
 4   Adjusted gross (in 2022 dollars)  20 non-null     object
 5   Artist                             20 non-null     object
 6   Tour title                         20 non-null     object
 7   Year(s)                            20 non-null     object
 8   Shows                              20 non-null     int64
 9   Average gross                      20 non-null     object
 10  Ref.                               20 non-null     object
dtypes: int64(2), object(9)
memory usage: 1.8+ KB
```

[ ]:
```python
df.dtypes
```

3.11 Handling Inconsistency

[8]:
```python
pattern = "(\[[0-9]*[a-z]*\])"
df.Peak = df.Peak.apply(lambda x: re.sub(pattern, "", str(x)))
df["All Time Peak"] = df["All Time Peak"].apply(lambda x: re.sub(pattern, "", str(x)))
df["Tour title"] = df["Tour title"].apply(lambda x: re.sub(pattern, "", str(x)))


df["Year_Start"] = df["Year(s)"].apply(lambda x: x[0:4])
df["Year_End"] = df["Year(s)"].apply(lambda x: x[-4:])
```

[9]:
```python
df
```

[9]:
```
    Rank Peak All Time Peak     Actual gross Adjusted gross (in 2022 dollars)  \
0      1    1             2     $780,000,000                     $780,000,000
1      2    1             7     $579,800,000                     $579,800,000
2      3    1             2     $411,000,000                     $560,622,615
3      4    2            10     $397,300,000                     $454,751,555
4      5    2           nan     $345,675,146                     $402,844,849
5      6    2            10     $305,158,363                     $388,978,496
6      7    2           nan     $280,000,000                     $381,932,682
7      7  nan           nan     $257,600,000                     $257,600,000
8      9  nan           nan     $256,084,556                     $312,258,401
9     10  nan           nan     $250,400,000                     $309,141,878
10    11  nan           nan  $229,100,000[b]                     $283,202,896
11    12  nan            14     $227,400,000                     $295,301,479
12    13  nan           nan     $204,000,000                     $251,856,802
13    14    1           nan     $200,000,000                     $299,676,265
14    15    2           nan     $194,000,000                     $281,617,035
15    16  nan           nan     $184,000,000                     $227,452,347
16    17  nan           nan     $170,000,000                     $213,568,571
17    18  nan           nan     $169,800,000                     $207,046,755
18    19  nan           nan  $167,700,000[e]                     $204,486,106
19    20  nan           nan     $150,000,000                     $185,423,109

          Artist                   Tour title    Year(s)  Shows  \
0   Taylor Swift               The Eras Tour †  2023–2024     56
1        Beyoncé       Renaissance World Tour       2023     56
2        Madonna         Sticky & Sweet Tour ‡  2008–2009     85
3           Pink   Beautiful Trauma World Tour  2018–2019    156
4   Taylor Swift      Reputation Stadium Tour       2018     53
5        Madonna                The MDNA Tour       2012     88
6     Celine Dion    Taking Chances World Tour  2008–2009    131
7           Pink              Summer Carnival †  2023–2024     41
8        Beyoncé        The Formation World Tour       2016     49
9   Taylor Swift           The 1989 World Tour       2015     85
```

```
10        Beyoncé    The Mrs. Carter Show World Tour  2013-2014    132
11      Lady Gaga            The Monster Ball Tour *  2009-2011    203
12     Katy Perry               Prismatic World Tour  2014-2015    151
13           Cher  Living Proof: The Farewell Tour ‡  2002-2005    325
14        Madonna                   Confessions Tour       2006     60
15           Pink           The Truth About Love Tour  2013-2014    142
16      Lady Gaga                  Born This Way Ball  2012-2013     98
17        Madonna                   Rebel Heart Tour  2015-2016     82
18          Adele                    Adele Live 2016  2016-2017    121
19   Taylor Swift                       The Red Tour  2013-2014     86

   Average gross       Ref. Year_Start Year_End
0    $13,928,571        [1]       2023     2024
1    $10,353,571        [3]       2023     2023
2     $4,835,294        [6]       2008     2009
3     $2,546,795        [7]       2018     2019
4     $6,522,173        [8]       2018     2018
5     $3,467,709        [9]       2012     2012
6     $2,137,405       [11]       2008     2009
7     $6,282,927       [12]       2023     2024
8     $5,226,215       [13]       2016     2016
9     $2,945,882       [14]       2015     2015
10    $1,735,606   [15][16]       2013     2014
11    $1,118,227       [18]       2009     2011
12    $1,350,993       [19]       2014     2015
13      $615,385       [20]       2002     2005
14    $3,233,333        [5]       2006     2006
15    $1,295,775       [22]       2013     2014
16    $1,734,694        [d]       2012     2013
17    $2,070,732        [4]       2015     2016
18    $1,385,950       [25]       2016     2017
19    $1,744,186       [26]       2013     2014
```

[10]:

```
DROPLIST = ["Ref.", "Year(s)"]
df.drop(DROPLIST, axis = 1, inplace= True)
print(df.columns)
```

```
Index(['Rank', 'Peak', 'All Time Peak', 'Actual gross',
       'Adjusted gross (in 2022 dollars)', 'Artist', 'Tour title', 'Shows',
       'Average gross', 'Year_Start', 'Year_End'],
      dtype='object')
```

[11]:

```
df.head()
```

[11]:

```
   Rank Peak All Time Peak  Actual gross Adjusted gross (in 2022 dollars)  \
0     1    1             2  $780,000,000                     $780,000,000
1     2    1             7  $579,800,000                     $579,800,000
```

```
2     3    1             2  $411,000,000                         $560,622,615
3     4    2            10  $397,300,000                         $454,751,555
4     5    2           nan  $345,675,146                         $402,844,849

         Artist                 Tour title  Shows  Average gross Year_Start  \
0  Taylor Swift             The Eras Tour †     56    $13,928,571       2023
1       Beyoncé      Renaissance World Tour     56    $10,353,571       2023
2       Madonna         Sticky & Sweet Tour ‡     85     $4,835,294       2008
3          Pink  Beautiful Trauma World Tour    156     $2,546,795       2018
4  Taylor Swift      Reputation Stadium Tour     53     $6,522,173       2018

  Year_End
0     2024
1     2023
2     2009
3     2019
4     2018
```

[12]:
```
for cols in df.columns:
    if df[cols].dtype == "object":
        df[cols] = df[cols].apply(lambda x: x.replace("$","").replace("nan",␣
 ↪"0"))
        df[cols] = df[cols].apply(lambda x: re.sub(pattern, "", str(x)))
    else:
        print(cols, df[cols].dtype)
```

```
Rank int64
Shows int64
```

[13]:
```
df["Peak"].astype(int)
df["All Time Peak"].astype(int)

for cols in [df.columns[3], df.columns[4], df.columns[7],df.columns[8],df.
 ↪columns[9],df.columns[10]]:
            df[cols] = df[cols].apply(lambda x: str(x).replace(",","").
 ↪replace(".", ""))
            df[cols] = df[cols].astype(int)

df.head()
```

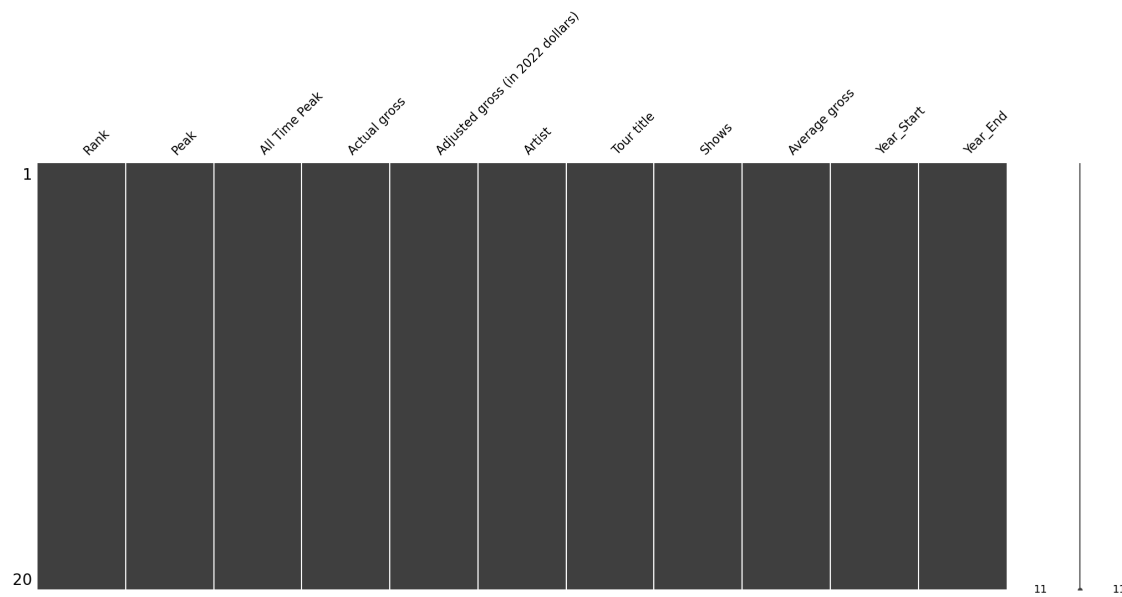
[13]:
```
   Rank Peak All Time Peak  Actual gross  Adjusted gross (in 2022 dollars)  \
0     1    1             2     780000000                         780000000
1     2    1             7     579800000                         579800000
2     3    1             2     411000000                         560622615
3     4    2            10     397300000                         454751555
4     5    2             0     345675146                         402844849
```

```
         Artist                    Tour title  Shows  Average gross  \
0  Taylor Swift              The Eras Tour †     56       13928571
1       Beyoncé        Renaissance World Tour     56       10353571
2       Madonna          Sticky & Sweet Tour ‡     85        4835294
3          Pink  Beautiful Trauma World Tour    156        2546795
4  Taylor Swift       Reputation Stadium Tour     53        6522173

   Year_Start  Year_End
0        2023      2024
1        2023      2023
2        2008      2009
3        2018      2019
4        2018      2018
```

[14]:
```python
msn.matrix(df)
```

[14]: <Axes: >



[15]:
```python
df.sort_values("Average gross", ascending = False).head(5)
```

[15]:
```
   Rank Peak All Time Peak  Actual gross  Adjusted gross (in 2022 dollars)  \
0     1    1             2     780000000                         780000000
1     2    1             7     579800000                         579800000
4     5    2             0     345675146                         402844849
7     7    0             0     257600000                         257600000
8     9    0             0     256084556                         312258401
```

```
         Artist                  Tour title  Shows  Average gross  Year_Start  \
0  Taylor Swift            The Eras Tour †     56       13928571        2023
1       Beyoncé    Renaissance World Tour     56       10353571        2023
4  Taylor Swift   Reputation Stadium Tour     53        6522173        2018
7          Pink          Summer Carnival †     41        6282927        2023
8       Beyoncé  The Formation World Tour     49        5226215        2016

   Year_End
0      2024
1      2023
4      2018
7      2024
8      2016
```

3.12 Conclusion

Data cleaning is a critical task in data science that helps ensure the accuracy and reliability of analysis and decision-making. Through data cleaning, errors can be removed, data quality can be improved, and the data can be made more accurate and complete.

```
[ ]:
```