Pandas Fundamentals

- Pandas is a powerful and open-source Python library. The Pandas library is used for data manipulation and analysis.
- Pandas consist of data structures and functions to perform efficient operations on data.
- The Pandas library is an essential tool for data analysts, scientists, and engineers working with structured data in Python.
- It is built on top of the NumPy library which means that a lot of the structures of NumPy are used or replicated in Pandas.
- The data produced by Pandas is often used as input for plotting functions in Matplotlib, statistical analysis in SciPy, and machine learning algorithms in Scikit-learn.
- Here is a list of things that we can do using Pandas.
  - Data set cleaning, merging, and joining.
  - Easy handling of missing data (represented as `NaN`) in floating point as well as non-floating-point data.
  - Columns can be inserted and deleted from DataFrame and higher-dimensional objects.
  - Powerful group by functionality for performing split-apply-combine operations on data sets.
  - Data Visualization.
- The Pandas library allows you to work with the following types of data:
  - Tabular data with columns of different data types, such as that from Excel spreadsheets, CSV files from the internet, and SQL database tables
  - Time series data, either at fixed-frequency or not
  - Other structured datasets, such as those coming from web data, like JSON files

Installing Pandas

- The first step in working with Pandas is to ensure whether it is installed in the system or not. If not, then we need to install it on our system using the `pip` command.
- Follow these steps to install Pandas:
  - **Step 1:** Type `'cmd'` in the search box and open it.
  - **Step 2:** Locate the folder using the `cd` command where the `python-pip file` has been installed.
  - **Step 3**: After locating it, type the command:

    ```
    pip install pandas
    ```

Importing Pandas

- After the Pandas have been installed in the system, you need to import the library. This module is generally imported as follows:

  ```
  import pandas as pd
  ```

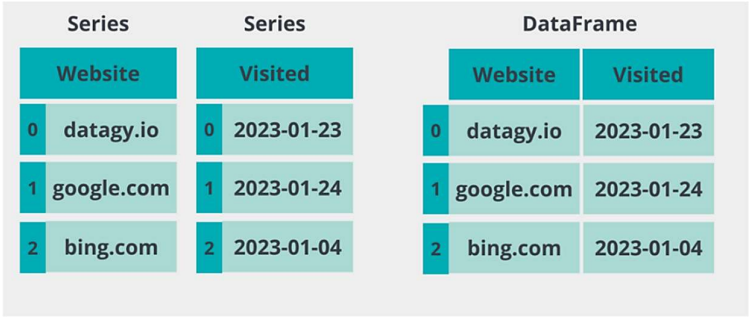
Data Structures in Pandas Library

- Pandas generally provide two data structures for manipulating data. They are:
  - **Series**

    The Pandas `Series` structure, is a one-dimensional **homogenous** array.

  - **DataFrame**

    The pandas `DataFrame` structure, is a two-dimensional, mutable, and potentially heterogeneous structure.



- Because the DataFrame is a container for the Series, they can also share a similar language for accessing, manipulating, and working with the data.
- Similarly, by providing two data structures, Pandas makes it much easier to work with two-dimensional data.

Creating Pandas DataFrames from Scratch

- To create a Pandas `DataFrame`, you can pass data directly into the `pd.DataFrame()` constructor.
- This allows you to pass in different types of Python data structures, such as `lists`, `dictionaries`, or `tuples`.
- **Loading a List of Tuples into a pandas `DataFrame`**

```
# Loading a List of Tuples into a Pandas DataFrame
import pandas as pd
data = [ ('Nik', 34, '2022-12-12'),
         ('Katie', 33, '2022-12-01'),
         ('Evan', 35, '2022-02-02'),
         ('Kyra', 34, '2022-04-14')]

df = pd.DataFrame(data)
print(df)
```

```
       0   1           2
0    Nik  34  2022-12-12
1  Katie  33  2022-12-01
2   Evan  35  2022-02-02
3   Kyra  34  2022-04-14
```

- we can add meaningful columns to the `DataFrame` by using the `columns=` parameter in the constructor.

```
# Loading a List of Tuples into a Pandas DataFrame
import pandas as pd
data = [ ('Nik', 34, '2022-12-12'),
         ('Katie', 33, '2022-12-01'),
         ('Evan', 35, '2022-02-02'),
         ('Kyra', 34, '2022-04-14')]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Date Joined'])
print(df)
```

```
    Name  Age Date Joined
0    Nik   34  2022-12-12
1  Katie   33  2022-12-01
2   Evan   35  2022-02-02
3   Kyra   34  2022-04-14
```

- **Loading a List of Dictionaries into a pandas `DataFrame`**

```
# Loading a List of Dictionaries into a Pandas DataFrame
import pandas as pd
data = [ {'Name': 'Nik', 'Age': 34, 'Date Joined': '2022-12-12'},
         {'Name': 'Katie', 'Age': 33, 'Date Joined': '2022-12-01'},
         {'Name': 'Evan', 'Age': 35, 'Date Joined': '2022-02-02'},
         {'Name': 'Kyra', 'Age': 34, 'Date Joined': '2022-04-14'} ]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Date Joined'])
print(df)
```

```
    Name  Age Date Joined
0    Nik   34  2022-12-12
1  Katie   33  2022-12-01
2   Evan   35  2022-02-02
3   Kyra   34  2022-04-14
```

- **Reading Data into Pandas `DataFrame`**
  - Pandas offers a lot of functionality for reading different data types into DataFrames.
  - For example, you can read data from Excel files, text files (like CSV), SQL databases, web APIs stored in JSON data, and even directly from webpages.
  - Pandas offers a number of custom functions for loading in different types of datasets:
    - The `pandas.read_csv()` function is used for reading CSV files into a pandas `DataFrame`
    - The `pandas.read_excel()` function is used for reading Excel files into a pandas `DataFrame`
    - The `pandas.read_html()` function is used for reading HTML tables into a pandas `DataFrame`
    - The `pandas.read_parquet()` function is used for reading parquet files into a pandas `DataFrame`
    - and many more…

```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df1 = pd.read_csv('https://raw.githubusercontent.com/datagy/data/main/data.csv')
print('Read Data1')
print(df1)
```

```
Read Data1
           Date Region                 Type  Units   Sales
0    2020-07-11   East  Children's Clothing   18.0   306.0
1    2020-09-23  North  Children's Clothing   14.0   448.0
2    2020-04-02  South     Women's Clothing   17.0   425.0
3    2020-02-28   East  Children's Clothing   26.0   832.0
4    2020-03-19   West     Women's Clothing    3.0    33.0
..          ...    ...                  ...    ...     ...
995  2020-02-11   East  Children's Clothing   35.0   735.0
996  2020-12-25  North       Men's Clothing    NaN  1155.0
997  2020-08-31  South       Men's Clothing   13.0   208.0
998  2020-08-23  South     Women's Clothing   17.0   493.0
999  2020-08-17  North     Women's Clothing   25.0   300.0

[1000 rows x 5 columns]
```

```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
print('\nRead Data2')
df2 = pd.read_csv('Data.csv')
print(df2)

```

```
Read Data2
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
```

- The `.head(n)` method returns the first n rows of a `DataFrame` (and defaults to five rows).
- The `.tail(n)` method returns the last n rows of a `DataFrame` (and defaults to five rows).

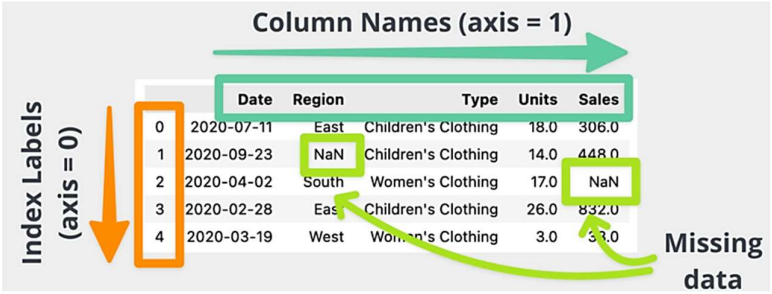
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('Data.csv')
print('Reading first 5 rows of data')
print(df.head())
print('\nReading first 5 rows of data')
print(df.tail())
```

```
Reading first 5 rows of data
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0

Reading first 5 rows of data
     Duration  Pulse  Maxpulse  Calories
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4
```

Let's take a look at the individual components of a pandas DataFrame:

- The `index` represents the "row" labels of a dataset.
  - In pandas, the index is also represented as the 0th axis. You can see how the index of our `DataFrame` above are bolded numbers from 0 through the end.
  - In this case the index labels are arbitrary, but can also represent unique, intelligible values.



- The columns are represented by the 1st axis of the `DataFrame` and are each made up of individual pandas Series objects.
  - Each column can have a unique data type, though joined together the `DataFrame` can be heterogeneous.
- The actual data can be of different data types, including dates. The data can also contain missing data, as represented by the `NaN` (not a number) values.

Understanding data using .describe()

- The `.describe()` method prints the summary statistics of all numeric columns, such as count, mean, standard deviation, range, and quartiles of numeric columns.

```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('Data.csv')

df.describe()

```

	Duration	Pulse	Maxpulse	Calories
count	169.000000	169.000000	169.000000	164.000000
mean	63.846154	107.461538	134.047337	375.800000
std	42.299949	14.510259	16.450434	266.377134
min	15.000000	80.000000	100.000000	50.300000
25%	45.000000	100.000000	124.000000	250.925000
50%	60.000000	105.000000	131.000000	318.600000
75%	60.000000	111.000000	141.000000	387.600000
max	300.000000	159.000000	184.000000	1860.400000

- You can also modify the quartiles using the percentiles argument. Here, for example, we're looking at the 30%, 50%, and 70% percentiles of the numeric columns in DataFrame df.

```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('Data.csv')

df.describe(percentiles=[0.3, 0.5, 0.7])
```

	Duration	Pulse	Maxpulse	Calories
count	169.000000	169.000000	169.000000	164.000000
mean	63.846154	107.461538	134.047337	375.800000
std	42.299949	14.510259	16.450434	266.377134
min	15.000000	80.000000	100.000000	50.300000
30%	45.000000	100.000000	125.400000	270.400000
50%	60.000000	105.000000	131.000000	318.600000
70%	60.000000	110.000000	138.600000	374.000000
max	300.000000	159.000000	184.000000	1860.400000

- In case of categorical data the `.describe()` method summarizes by number of observations, number of unique elements, mode, and frequency of the mode.

```python
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/datagy/data/main/data.csv')
print(df.head())
print('\nDescribing Regions')
print(df['Region'].describe())
```

```
         Date Region                Type  Units  Sales
0  2020-07-11   East  Children's Clothing   18.0  306.0
1  2020-09-23  North  Children's Clothing   14.0  448.0
2  2020-04-02  South     Women's Clothing   17.0  425.0
3  2020-02-28   East  Children's Clothing   26.0  832.0
4  2020-03-19   West     Women's Clothing    3.0   33.0

Describing Regions
count     1000
unique       4
top       East
freq       411
Name: Region, dtype: object
```

Understanding data using .info()

- The `.info()` method is a quick way to look at the data types, missing values, and data size of a DataFrame.
    - `show_counts` = `True`: gives a few over the total non-missing values in each column.
    - `memory_usage = True`: shows the total memory usage of the DataFrame elements.
    - `verbose = True`: prints the full summary from `.info()`.

```python
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('Data.csv')

df.info(show_counts=True, memory_usage=True, verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

Understanding your data using .shape

- The number of rows and columns of a DataFrame can be identified using the `.shape` attribute of the DataFrame.
- It returns a tuple (row, column) and can be indexed to get only rows, and only columns count as output.

```python
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')

print(df.shape)
print(df.shape[0])
print(df.shape[1])

```

```
(768, 9)
768
9
```

Get all columns and column names

- Calling the `.columns` attribute of a DataFrame object returns the column names in the form of an Index object. As a reminder, a pandas index is the address/label of the row or column.
- This can also be converted to a Python list object.

```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')

print(df.columns)
list(df.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')

['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age',
 'Outcome']
```

Accessing a DataFrame's Columns

```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93],
                      'Eva': [95, 99, 87],
                      'Sam': [88, 94, 85]},
                     index= ['Physics', 'Chemistry', 'Mathematics'])
print('All marks:')
print(marks)
print('\nEva\'s marks:')
print(marks['Eva'])
print('\nWally\'s marks:')
print(marks.Wally)
```

```
All marks:
             Wally  Eva  Sam
Physics         87   95   88
Chemistry       89   99   94
Mathematics     93   87   85

Eva's marks:
Physics        95
Chemistry      99
Mathematics    87
Name: Eva, dtype: int64

Wally's marks:
Physics        87
Chemistry      89
Mathematics    93
Name: Wally, dtype: int64
```

Selecting Rows via the loc and iloc Attributes

```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])
print('All marks:')
print(marks)

print('\nSelecting Test01 marks using loc')
print(marks.loc['Test01'])
print('\nSelecting Test01 marks using iloc')
print(marks.iloc[0])
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
Test04     87   88   89     84   83
Test05     92   84   95     85   87

Selecting Test01 marks using loc
Wally    87
Eva      95
Sam      88
Katie    87
Bob      83
Name: Test01, dtype: int64

Selecting Test01 marks using iloc
Wally    87
Eva      95
Sam      88
Katie    87
Bob      83
Name: Test01, dtype: int64
```

Selecting Rows via Slices and Lists with the loc and iloc Attributes

```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])

print(marks.loc['Test01':'Test02'])
print(marks.iloc[0:2])
print()
print(marks.loc[['Test01','Test03']])
print(marks.iloc[[0,2]])
```

```
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93

        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test03     93   87   85     95   86
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test03     93   87   85     95   86
```

Boolean Indexing

```python
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])
print('O grade students')
print(marks[marks>=90]) # O grade students
print('\nA grade students')
print(marks[(marks>=80) & (marks<90)]) # A grade students
```

```
O grade students
        Wally   Eva   Sam  Katie   Bob
Test01    NaN  95.0   NaN    NaN   NaN
Test02    NaN  99.0  94.0   92.0  93.0
Test03   93.0   NaN   NaN   95.0   NaN
Test04    NaN   NaN   NaN    NaN   NaN
Test05   92.0   NaN  95.0    NaN   NaN

A grade students
        Wally   Eva   Sam  Katie   Bob
Test01   87.0   NaN  88.0   87.0  83.0
Test02   89.0   NaN   NaN    NaN   NaN
Test03    NaN  87.0  85.0    NaN  86.0
Test04   87.0  88.0  89.0   84.0  83.0
Test05    NaN  84.0   NaN   85.0  87.0
```

Accessing a Specific DataFrame Cell by Row and Column

```python
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])
print('All marks:')
print(marks)

print('\nEva\'s Test01 marks:')
print(marks['Eva']['Test01'])      # Column name then Index name
print(marks.Eva.Test01)            # Column name then Index name
print(marks.at['Test01', 'Eva'])   # Row name then Column name
print(marks.iat[0, 1])             # Row index then Column index
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
Test04     87   88   89     84   83
Test05     92   84   95     85   87

Eva's Test01 marks:
95
95
95
95
```

Selecting Selected Rows and Columns

```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])
print('All marks:')
print(marks)
print('\nSelect all rows of Col1')
print(marks.iloc[:, 0])
print('\nSelect all columns of Row1')
print(marks.iloc[0, :])
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
Test04     87   88   89     84   83
Test05     92   84   95     85   87

Select all rows of Col1
Test01    87
Test02    89
Test03    93
Test04    87
Test05    92
Name: Wally, dtype: int64

Select all columns of Row1
Wally    87
Eva      95
Sam      88
Katie    87
Bob      83
Name: Test01, dtype: int64
```

Transposing the DataFrame with the T Attribute

```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95]},
                      index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test05'])
marksTransposed = marks.T
print('All marks:')
print(marks)
print('All marks Transposed:')
print(marksTransposed)
```

```
All marks:
        Wally  Eva  Sam
Test01     87   95   88
Test02     89   99   94
Test03     93   87   85
Test04     87   88   89
Test05     92   84   95
All marks Transposed:
       Test01  Test02  Test03  Test04  Test05
Wally      87      89      93      87      92
Eva        95      99      87      88      84
Sam        88      94      85      89      95
```

Sorting by Rows and Columns by Their Indices (axis=0 for rows and axis = 1 for columns)

```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93],
                      'Eva': [95, 99, 87],
                      'Sam': [88, 94, 85],
                      'Katie': [87, 92, 95],
                      'Bob': [83, 93, 86]},
                      index = ['Test01', 'Test02', 'Test03'])
print('All marks:')
print(marks)
print('All marks Sorted by Row indices:')
print(marks.sort_index(ascending=False))
print('All marks Sorted by Column indices:')
print(marks.sort_index(axis=1))
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
All marks Sorted by Row indices:
        Wally  Eva  Sam  Katie  Bob
Test03     93   87   85     95   86
Test02     89   99   94     92   93
Test01     87   95   88     87   83
All marks Sorted by Column indices:
        Bob  Eva  Katie  Sam  Wally
Test01   83   95     87   88     87
Test02   93   99     92   94     89
Test03   86   87     95   85     93
```

Sorting by Column Values (axis = 1 for columns)

```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93],
                      'Eva': [95, 99, 87],
                      'Sam': [88, 94, 85],
                      'Katie': [87, 92, 95],
                      'Bob': [83, 93, 86]},
                      index = ['Test01', 'Test02', 'Test03'])
print('All marks:')
print(marks)
print('\nAll marks Sorted by Column values:')
print(marks.sort_values(by='Test01', axis=1, ascending=False))
print(marks.sort_values(by='Test02', axis=1, ascending=False))
print('\nTranspose and sort:')
print(marks.T.sort_values(by='Test01', ascending=False))
print('\nSelect and sort:')
print(marks.loc['Test01'].sort_values(ascending=False))
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86

All marks Sorted by Column values:
        Eva  Sam  Wally  Katie  Bob
Test01   95   88     87     87   83
Test02   99   94     89     92   93
Test03   87   85     93     95   86
```

```
        Eva  Sam  Bob  Katie  Wally
Test01   95   88   83     87     87
Test02   99   94   93     92     89
Test03   87   85   86     95     93

Transpose and sort:
       Test01  Test02  Test03
Eva        95      99      87
Sam        88      94      85
Wally      87      89      93
Katie      87      92      95
Bob        83      93      86

Select and sort:
Eva      95
Sam      88
Wally    87
Katie    87
Bob      83
Name: Test01, dtype: int64
```

Handling Duplicates

Method	Objective
`df.drop_duplicates()`	will return a copy of your DataFrame, with duplicates removed
`df.drop_duplicates(inplace=True)`	will modify the DataFrame object in place, with duplicates removed
`df.drop_duplicates(inplace=True, keep=first)`	Drop duplicates in place except for the first occurrence.
`df.drop_duplicates(inplace=True, keep=last)`	Drop duplicates in place except for the last occurrence.
`df.drop_duplicates(inplace=True, keep=False)`	Drop all duplicates.

Column cleanup

- Many times, datasets will have verbose column names with symbols, upper and lowercase words, spaces, and typos.
- To make selecting data by column name easier we can spend a little time cleaning up their names.

```
df.rename(columns={
        'Runtime (Minutes)': 'Runtime_mins',
        'Revenue (Millions)': 'Revenue_millions'
    }, inplace=True)
```

- In order to lowercase all names of columns we can use the `col.lower()` method as follows.

```
df.columns = [col.lower() for col in df]
```

How to work with missing values

- When exploring data, you'll most likely encounter missing or null values, which are essentially placeholders for non-existent values. Most commonly you'll see Python's `None` or NumPy's `np.nan`, each of which are handled differently in some situations.
- There are two options in dealing with nulls:
  - Get rid of rows or columns with nulls
  - Replace nulls with non-null values, a technique known as imputation.
    - Imputation is a conventional feature engineering technique used to keep valuable data that have null values.
    - There may be instances where dropping every row with a null value removes too big a chunk from your dataset, so instead we can impute that null with another value, usually the **mean** or the **median** of that column.

Method	Objective
`df.isnull()`	• Checks which cells in our DataFrame are null • Returns a DataFrame where each cell is either True or False depending on that cell's null status.
`df.isnull().sum()`	• Counts the number of nulls in each column
`df.dropna()`	• Removes the null valued rows
`df.dropna(axis=1)`	• Removes the null valued columns
`df['ColName'] = df['ColName'].fillna(value)`	• Replace null values with value
`value = df['ColName'].mean()` `df['ColName'] = df['ColName'].fillna(value)`	• Replace null values with mean value
`value = df['ColName'].median()` `df['ColName'] = df['ColName'].fillna(value)`	• Replace null values with median value
`df.ffill() or df.pad()`	• Uses the previous value to fill missing values in a gap
`df.bfill()`	• Uses the next value to fill missing values in a gap

In [2]:
```
# Loading a List of Tuples into a Pandas DataFrame
import pandas as pd
data = [ ('Nik', 34, '2022-12-12'),
         ('Katie', 33, '2022-12-01'),
         ('Evan', 35, '2022-02-02'),
         ('Kyra', 34, '2022-04-14')]

df = pd.DataFrame(data)
print(df)
```

```
      0   1           2
0    Nik  34  2022-12-12
1  Katie  33  2022-12-01
2   Evan  35  2022-02-02
3   Kyra  34  2022-04-14
```

In [3]:
```
# Loading a List of Tuples into a Pandas DataFrame
import pandas as pd
data = [ ('Nik', 34, '2022-12-12'),
         ('Katie', 33, '2022-12-01'),
         ('Evan', 35, '2022-02-02'),
         ('Kyra', 34, '2022-04-14')]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Date Joined'],
                  index = ['Student1', 'Student2','Student3','Student4'])
print(df)
```

```
           Name  Age Date Joined
Student1    Nik   34  2022-12-12
Student2  Katie   33  2022-12-01
Student3   Evan   35  2022-02-02
Student4   Kyra   34  2022-04-14
```

In [4]:
```
# Loading a List of Dictionaries into a Pandas DataFrame
import pandas as pd
data = [ {'Name': 'Nik', 'Age': 34, 'Date Joined': '2022-12-12'},
         {'Name': 'Katie', 'Age': 33, 'Date Joined': '2022-12-01'},
         {'Name': 'Evan', 'Age': 35, 'Date Joined': '2022-02-02'},
         {'Name': 'Kyra', 'Age': 34, 'Date Joined': '2022-04-14'} ]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Date Joined'])
print(df)
```

```
    Name  Age Date Joined
0    Nik   34  2022-12-12
1  Katie   33  2022-12-01
2   Evan   35  2022-02-02
3   Kyra   34  2022-04-14
```

In [5]:
```
import pandas as pd
data =  {'Name': ('Nik', 'Katie','Evan','Kyra'),
         'Age': [34, 35, 36, 37],
         'Date Joined': ['2022-12-12', '2022-12-01','2022-02-02','2022-04-14
```

```
df = pd.DataFrame(data)
print(df)
```

```
   Name  Age Date Joined
0   Nik   34  2022-12-12
1  Katie   35  2022-12-01
2   Evan   36  2022-02-02
3   Kyra   37  2022-04-14
```

In [6]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df1 = pd.read_csv('https://raw.githubusercontent.com/datagy/data/main/data.c
print('Read Data1')
print(df1)
```

```
Read Data1
          Date Region                 Type  Units   Sales
0    2020-07-11   East  Children's Clothing   18.0   306.0
1    2020-09-23  North  Children's Clothing   14.0   448.0
2    2020-04-02  South     Women's Clothing   17.0   425.0
3    2020-02-28   East  Children's Clothing   26.0   832.0
4    2020-03-19   West     Women's Clothing    3.0    33.0
..          ...    ...                  ...    ...     ...
995  2020-02-11   East  Children's Clothing   35.0   735.0
996  2020-12-25  North       Men's Clothing    NaN  1155.0
997  2020-08-31  South       Men's Clothing   13.0   208.0
998  2020-08-23  South     Women's Clothing   17.0   493.0
999  2020-08-17  North     Women's Clothing   25.0   300.0

[1000 rows x 5 columns]
```

In [7]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
print('\nRead Data2')
df2 = pd.read_csv('diabetes.csv')
print(df2)
```

```
Read Data2
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

In [8]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')
print('Reading first 5 rows of data')
print(df.head(10))
print('\nReading first 5 rows of data')
print(df.tail())
```

```
Reading first 5 rows of data
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1
5            5      116             74              0        0  25.6
6            3       78             50             32       88  31.0
7           10      115              0              0        0  35.3
8            2      197             70             45      543  30.5
9            8      125             96              0        0   0.0

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
8                     0.158   53        1
9                     0.232   54        1

Reading first 5 rows of data
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0
```

In [9]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')

df.describe()
```

Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [10]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')

df.describe(percentiles=[0.3, 0.5, 0.7])
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
30%	1.000000	102.000000	64.000000	8.200000	0.000000	28.200000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
70%	5.000000	134.000000	78.000000	31.000000	106.000000	35.490000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [11]:
```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/datagy/data/main/data.cs
print(df.head())
print(df.describe())
print('\nDescribing Regions')
print(df['Region'].describe())
print('\nDescribing Type')
print(df['Type'].describe())
print('\nDescribing Date')
print(df['Date'].describe())
```

```
         Date Region                 Type  Units  Sales
0  2020-07-11   East  Children's Clothing   18.0  306.0
1  2020-09-23  North  Children's Clothing   14.0  448.0
2  2020-04-02  South     Women's Clothing   17.0  425.0
3  2020-02-28   East  Children's Clothing   26.0  832.0
4  2020-03-19   West     Women's Clothing    3.0   33.0
            Units        Sales
count  911.000000  1000.000000
mean    19.638858   427.254000
std      9.471309   253.441362
min      3.000000    33.000000
25%     12.000000   224.000000
50%     20.000000   380.000000
75%     28.000000   575.000000
max     35.000000  1155.000000

Describing Regions
count     1000
unique       4
top       East
freq       411
Name: Region, dtype: object

Describing Type
count                 1000
unique                   3
top       Women's Clothing
freq                   424
Name: Type, dtype: object

Describing Date
count           1000
unique           347
top       2020-12-17
freq              10
Name: Date, dtype: object
```

In [12]:
```python
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')

df.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.0
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.2
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.0
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.0
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.2
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.6
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.6
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.0
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.0

In [13]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [14]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
df = pd.read_csv('diabetes.csv')

print(df.shape)
print(df.shape[0])
print(df.shape[1])
```

```
(768, 9)
768
9
```

In [15]:
```
# Loading a CSV File into a Pandas DataFrame
import pandas as pd
```

```
df = pd.read_csv('diabetes.csv')

print(df.columns)
list(df.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

Out[15]:
```
['Pregnancies',
 'Glucose',
 'BloodPressure',
 'SkinThickness',
 'Insulin',
 'BMI',
 'DiabetesPedigreeFunction',
 'Age',
 'Outcome']
```

In [16]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93],
                      'Eva': [95, 99, 87],
                      'Sam': [88, 94, 85]},
                     index= ['Physics', 'Chemistry', 'Mathematics'])
print('All marks:')
print(marks)
print('\nEva\'s marks:')
print(marks['Eva'])
print('\nWally\'s marks:')
print(marks.Wally)
```

```
All marks:
             Wally  Eva  Sam
Physics         87   95   88
Chemistry       89   99   94
Mathematics     93   87   85

Eva's marks:
Physics        95
Chemistry      99
Mathematics    87
Name: Eva, dtype: int64

Wally's marks:
Physics        87
Chemistry      89
Mathematics    93
Name: Wally, dtype: int64
```

In [17]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93],
                      'Eva': [95, 99, 87],
                      'Sam': [88, 94, 85]},
                     index= ['Physics', 'Chemistry', 'Mathematics'])
```

```
print('All marks:')
print(marks)
print('\nEva\'s phyics marks:')
print(marks['Eva']['Physics'])
print('\nWally\'s chemistry marks:')
print(marks.Wally.Chemistry)
```

```
All marks:
             Wally  Eva  Sam
Physics         87   95   88
Chemistry       89   99   94
Mathematics     93   87   85

Eva's phyics marks:
95

Wally's chemistry marks:
89
```

In [18]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test0
print('All marks:')
print(marks)

print('\nSelecting Test01 marks using loc')
print(marks.loc['Test01'])
print('\nSelecting Test01 marks using iloc')
print(marks.iloc[0])
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
Test04     87   88   89     84   83
Test05     92   84   95     85   87

Selecting Test01 marks using loc
Wally    87
Eva      95
Sam      88
Katie    87
Bob      83
Name: Test01, dtype: int64

Selecting Test01 marks using iloc
Wally    87
Eva      95
Sam      88
Katie    87
Bob      83
Name: Test01, dtype: int64
```

In [19]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test0

print(marks.loc['Test01':'Test02'])
print(marks.iloc[0:2])
print()
print(marks.loc[['Test01','Test03']])
print(marks.iloc[[0,2]])
```

```
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93

        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test03     93   87   85     95   86
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test03     93   87   85     95   86
```

In [20]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
```

```
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test0
print(marks>=90)
print('O grade students')
print(marks[marks>=90]) # O grade students
print('\nA grade students')
print(marks[(marks>=80) & (marks<90)]) # A grade students
```

```
       Wally    Eva    Sam  Katie    Bob
Test01  False   True  False  False  False
Test02  False   True   True   True   True
Test03   True  False  False   True  False
Test04  False  False  False  False  False
Test05   True  False   True  False  False
O grade students
       Wally   Eva   Sam  Katie   Bob
Test01   NaN  95.0   NaN    NaN   NaN
Test02   NaN  99.0  94.0   92.0  93.0
Test03  93.0   NaN   NaN   95.0   NaN
Test04   NaN   NaN   NaN    NaN   NaN
Test05  92.0   NaN  95.0    NaN   NaN

A grade students
       Wally   Eva   Sam  Katie   Bob
Test01  87.0   NaN  88.0   87.0  83.0
Test02  89.0   NaN   NaN    NaN   NaN
Test03   NaN  87.0  85.0    NaN  86.0
Test04  87.0  88.0  89.0   84.0  83.0
Test05   NaN  84.0   NaN   85.0  87.0
```

In [21]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test0
print('All marks:')
print(marks)

print('\nEva\'s Test01 marks:')
print(marks['Eva']['Test01'])      # Column name then Index name
print(marks.Eva.Test01)            # Column name then Index name
print(marks.at['Test01', 'Eva'])   # Row name then Column name
print(marks.iat[0, 1])             # Row index then Column index
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
Test04     87   88   89     84   83
Test05     92   84   95     85   87

Eva's Test01 marks:
95
95
95
95
```

In [22]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95],
                      'Katie': [87, 92, 95, 84, 85],
                      'Bob': [83, 93, 86, 83, 87]},
                     index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test0
print('All marks:')
print(marks)
print('\nSelect all rows of Col1')
print(marks.iloc[:, 0])
print('\nSelect all columns of Row1')
print(marks.iloc[0, :])
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
Test04     87   88   89     84   83
Test05     92   84   95     85   87

Select all rows of Col1
Test01    87
Test02    89
Test03    93
Test04    87
Test05    92
Name: Wally, dtype: int64

Select all columns of Row1
Wally    87
Eva      95
Sam      88
Katie    87
Bob      83
Name: Test01, dtype: int64
```

In [23]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93, 87, 92],
```

```
                      'Eva': [95, 99, 87, 88, 84],
                      'Sam': [88, 94, 85, 89, 95]},
                      index = ['Test01', 'Test02', 'Test03', 'Test04', 'Test
marksTransposed = marks.T
print('All marks:')
print(marks)
print('All marks Transposed:')
print(marksTransposed)
```

```
All marks:
        Wally  Eva  Sam
Test01     87   95   88
Test02     89   99   94
Test03     93   87   85
Test04     87   88   89
Test05     92   84   95
All marks Transposed:
       Test01  Test02  Test03  Test04  Test05
Wally      87      89      93      87      92
Eva        95      99      87      88      84
Sam        88      94      85      89      95
```

In [24]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93],
                      'Eva': [95, 99, 87],
                      'Sam': [88, 94, 85],
                      'Katie': [87, 92, 95],
                      'Bob': [83, 93, 86]},
                      index = ['Test01', 'Test02', 'Test03'])
print('All marks:')
print(marks)
print('All marks Sorted by Row indices:')
print(marks.sort_index(ascending=False))
print('All marks Sorted by Column indices:')
print(marks.sort_index(axis=1))
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86
All marks Sorted by Row indices:
        Wally  Eva  Sam  Katie  Bob
Test03     93   87   85     95   86
Test02     89   99   94     92   93
Test01     87   95   88     87   83
All marks Sorted by Column indices:
        Bob  Eva  Katie  Sam  Wally
Test01   83   95     87   88     87
Test02   93   99     92   94     89
Test03   86   87     95   85     93
```

In [25]:
```
import pandas as pd

marks = pd.DataFrame({'Wally': [87, 89, 93],
                      'Eva': [95, 99, 87],
```

```
                      'Sam': [88, 94, 85],
                      'Katie': [87, 92, 95],
                      'Bob': [83, 93, 86]},
                      index = ['Test01', 'Test02', 'Test03'])
print('All marks:')
print(marks)
print('\nAll marks Sorted by Column values:')
print(marks.sort_values(by='Test01', axis=1, ascending=False))
print(marks.sort_values(by='Test02', axis=1, ascending=False))
print('\nTranspose and sort:')
print(marks.T.sort_values(by='Test01', ascending=False))
print('\nSelect and sort:')
print(marks.loc['Test01'].sort_values(ascending=False))
```

```
All marks:
        Wally  Eva  Sam  Katie  Bob
Test01     87   95   88     87   83
Test02     89   99   94     92   93
Test03     93   87   85     95   86

All marks Sorted by Column values:
        Eva  Sam  Wally  Katie  Bob
Test01   95   88     87     87   83
Test02   99   94     89     92   93
Test03   87   85     93     95   86
        Eva  Sam  Bob  Katie  Wally
Test01   95   88   83     87     87
Test02   99   94   93     92     89
Test03   87   85   86     95     93

Transpose and sort:
       Test01  Test02  Test03
Eva        95      99      87
Sam        88      94      85
Wally      87      89      93
Katie      87      92      95
Bob        83      93      86

Select and sort:
Eva      95
Sam      88
Wally    87
Katie    87
Bob      83
Name: Test01, dtype: int64
```

In [26]:

```
import pandas as pd

movies_df = pd.read_csv("IMDBData.csv", index_col="Title")
movies_df.head()
```

Out[26]:

Title	Rank	Genre	Description	Director	Actors	Y
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2(
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	2(
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	2(
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	2(
Suicide Squad	5	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	2(

In [27]:
```
movies_df.tail()
```

Out[27]:

Title	Rank	Genre	Description	Director	Actors	Year	Run (Min
Secret in Their Eyes	996	Crime,Drama,Mystery	A tight-knit team of rising investigators, alo...	Billy Ray	Chiwetel Ejiofor, Nicole Kidman, Julia Roberts...	2015	
Hostel: Part II	997	Horror	Three American college students studying abroa...	Eli Roth	Lauren German, Heather Matarazzo, Bijou Philli...	2007	
Step Up 2: The Streets	998	Drama,Music,Romance	Romantic sparks occur between two dance studen...	Jon M. Chu	Robert Hoffman, Briana Evigan, Cassie Ventura,...	2008	
Search Party	999	Adventure,Comedy	A pair of friends embark on a mission to reuni...	Scot Armstrong	Adam Pally, T.J. Miller, Thomas Middleditch,Sh...	2014	
Nine Lives	1000	Comedy,Family,Fantasy	A stuffy businessman finds himself trapped ins...	Barry Sonnenfeld	Kevin Spacey, Jennifer Garner, Robbie Amell,Ch...	2016	

In [28]:
```
movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Rank                1000 non-null   int64  
 1   Genre               1000 non-null   object 
 2   Description         1000 non-null   object 
 3   Director            1000 non-null   object 
 4   Actors              1000 non-null   object 
 5   Year                1000 non-null   int64  
 6   Runtime (Minutes)   1000 non-null   int64  
 7   Rating              1000 non-null   float64
 8   Votes               1000 non-null   int64  
 9   Revenue (Millions)  872 non-null    float64
 10  Metascore           936 non-null    float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

In [29]:
```
movies_df.shape
```

Out[29]:
```
(1000, 11)
```

In [31]:
```
# Create duplicates
temp_df = pd.concat([movies_df,movies_df])

temp_df.shape
```

Out[31]:
```
(2000, 11)
```

In [32]:
```
# Remove duplicates
temp_df = temp_df.drop_duplicates()

temp_df.shape
```

Out[32]:
```
(1000, 11)
```

In [33]:
```
movies_df.columns
```

Out[33]:
```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)',
       'Metascore'],
      dtype='object')
```

In [34]:
```
movies_df.rename(columns={
        'Runtime (Minutes)': 'Runtime_mins',
        'Revenue (Millions)': 'Revenue_millions'
    }, inplace=True)


movies_df.columns
```

Out[34]:
```
Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year',
       'Runtime_mins', 'Rating', 'Votes', 'Revenue_millions', 'Metascore'],
      dtype='object')
```

In [35]:
```
movies_df.columns = [col.lower() for col in movies_df]

movies_df.columns
```

Out[35]:
```
Index(['rank', 'genre', 'description', 'director', 'actors', 'year',
       'runtime_mins', 'rating', 'votes', 'revenue_millions', 'metascore'],
      dtype='object')
```

In [36]:
```
movies_df.head()
```

Out[36]:

Title	rank	genre	description	director	actors	ye
Guardians of the Galaxy	1	Action,Adventure,Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	20
Prometheus	2	Adventure,Mystery,Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael Fa...	20
Split	3	Horror,Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richar...	20
Sing	4	Animation,Comedy,Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey,Reese Witherspoon, Seth Ma...	20
Suicide Squad	5	Action,Adventure,Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola D...	20

In [37]:
```
movies_df.isnull()
```

Out[37]:

Title	rank	genre	description	director	actors	year	runtime_mins	rating	votes
Guardians of the Galaxy	False	False	False	False	False	False	False	False	False
Prometheus	False	False	False	False	False	False	False	False	False
Split	False	False	False	False	False	False	False	False	False
Sing	False	False	False	False	False	False	False	False	False
Suicide Squad	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
Secret in Their Eyes	False	False	False	False	False	False	False	False	False
Hostel: Part II	False	False	False	False	False	False	False	False	False
Step Up 2: The Streets	False	False	False	False	False	False	False	False	False
Search Party	False	False	False	False	False	False	False	False	False
Nine Lives	False	False	False	False	False	False	False	False	False

1000 rows × 11 columns

In [38]:
```
movies_df.isnull().sum()
```

Out[38]:
```
rank                  0
genre                 0
description           0
director              0
actors                0
year                  0
runtime_mins          0
rating                0
votes                 0
revenue_millions    128
metascore            64
dtype: int64
```

In [40]:
```
movies_df['revenue_millions'] = movies_df['revenue_millions'].fillna(movies_
movies_df['metascore'] = movies_df['metascore'].fillna(movies_df['metascore'

movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   rank              1000 non-null   int64  
 1   genre             1000 non-null   object 
 2   description       1000 non-null   object 
 3   director          1000 non-null   object 
 4   actors            1000 non-null   object 
 5   year              1000 non-null   int64  
 6   runtime_mins      1000 non-null   int64  
 7   rating            1000 non-null   float64
 8   votes             1000 non-null   int64  
 9   revenue_millions  1000 non-null   float64
 10  metascore         1000 non-null   float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

In [46]:

```python
import pandas as pd

# Yet again, our starting point DataFrame
data = {"Name": ["John", "Jane", "Anna"],
        "Age": [28, None, 22],
        "City": [None, "New York", "London"]}
df = pd.DataFrame(data)
print(df)
print()

# Using 'ffill' to forward fill the missing values
df.ffill(inplace=True)
print(df)
print()

# For the sake of illustration, let's reset and use 'bfill'
df = pd.DataFrame(data)
df.bfill(inplace=True)
print(df)
```

```
   Name   Age      City
0  John  28.0      None
1  Jane   NaN  New York
2  Anna  22.0    London

   Name   Age      City
0  John  28.0      None
1  Jane  28.0  New York
2  Anna  22.0    London

   Name   Age      City
0  John  28.0  New York
1  Jane  22.0  New York
2  Anna  22.0    London
```