Pandas Handbook for Ai

Ashraful Islam Mahi

Full Stack Python Developer | ML & Robotics Enthusiast

What is Pandas?

- Pandas is a powerful python library for data manipulation and analysis. It provides an easy to use data structure which are series & dataframes.

Pandas Data Structures:

1. Series: 1D labeled array, capable of holding data of any type.
2. Datframe: 2D labeled data structure like table.

0. Installing pandas

In [ ]:
```
%pip install pandas
```

1. Importing Pandas

In [104…
```
import pandas as pd
```

2. Creating Series in Pandas

In [5]:
```
s = pd.Series([10,20,30],index=['a','b','c'])

print(f"Series:\n{s}")
```

```
Series:
a    10
b    20
c    30
dtype: int64
```

3. Creating Dataframe in Pandas

In [8]:
```
data = {'Name':['Alice','Bob'],'Age':[20,30]}

df = pd.DataFrame(data)
print(f"Dataframe:\n\n{df}")
```

```
Dataframe:

    Name  Age
0  Alice   20
1    Bob   30
```

Data loading Method in Methods in Pandas

- Common Data loading Methods:
  1. From CSV
  2. From Excel
  3. From Dictionary

1. Data loading from CSV

- students.csv :

Name	Math	Science	English
Alice	85	90	88
Bob	75	80	72
Charlie	95	98	97

In [12]:
```
df = pd.read_csv("students.csv")

print(f"Students Info:\n\n{df}")
```

```
Students Info:

      Name  Math  Science  English
0    Alice    85       90       88
1      Bob    75       80       72
2  Charlie    95       98       97
```

Data Saving Methods in Pandas

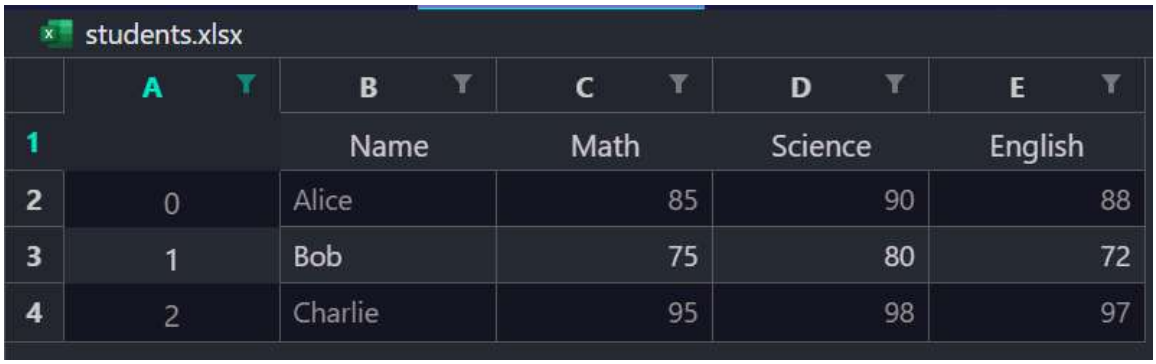
- It is similar to loading methods:
    1. In CSV
    2. In Excel
    3. In Dictionary

1. Data Saving in Excel

In [ ]:
```
%pip install openpyxl #mandatory to save data in excel
```

In [15]:
```
df.to_excel("students.xlsx")
```

- students.xlsx :



students.xlsx

	A	B	C	D	E
1		Name	Math	Science	English
2	0	Alice	85	90	88
3	1	Bob	75	80	72
4	2	Charlie	95	98	97

Dataframe Operations

1. Methods of Viewing Data

1. a) View the 1st row:

In [127…
```
df = pd.read_excel("students.xlsx")

print(f"1st row:\n\n{df.head(1)}")
```

```
1st row:

   Unnamed: 0   Name  Math  Science  English
0           0  Alice    85       90       88
```

1. b) View the last row:

In [128…
```
df = pd.read_excel("students.xlsx")
print(df.tail(1))
```

```
   Unnamed: 0     Name  Math  Science  English
2           2  Charlie    95       98       97
```

1. c) View dataframe info:

In [129…
```
df = pd.read_excel("students.xlsx")
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  3 non-null      int64 
 1   Name        3 non-null      object
 2   Math        3 non-null      int64 
 3   Science     3 non-null      int64 
 4   English     3 non-null      int64 
dtypes: int64(4), object(1)
memory usage: 252.0+ bytes
None
```

1. d) View dataframe description:

In [130…
```
df = pd.read_excel("students.xlsx")
print(df.describe())
```

```
       Unnamed: 0  Math    Science    English
count         3.0   3.0   3.000000   3.000000
mean          1.0  85.0  89.333333  85.666667
std           1.0  10.0   9.018500  12.662280
min           0.0  75.0  80.000000  72.000000
25%           0.5  80.0  85.000000  80.000000
50%           1.0  85.0  90.000000  88.000000
75%           1.5  90.0  94.000000  92.500000
max           2.0  95.0  98.000000  97.000000
```

2. Selecting & Indexing

In [131…
```
#Show the data from only Name & Math column
df[["Name","Math"]]
```

Out[131…

	Name	Math
0	Alice	85
1	Bob	75
2	Charlie	95

3. Filtering

In [132…
```
#Condtion: Show the row in which the mark in mathematics is above 80
df[df["Math"]>80]
```

Out[132…

	Unnamed: 0	Name	Math	Science	English
0	0	Alice	85	90	88
2	2	Charlie	95	98	97

4. Selecting by position

In [133…
```
df.iloc[0] #1st row by position
```

Out[133…
```
Unnamed: 0        0
Name          Alice
Math             85
Science          90
English          88
Name: 0, dtype: object
```

5. Selecting by label

In [134…
```
df.loc[:,"Math"] #showing the column named "Math"
```

Out[134…
```
0    85
1    75
2    95
Name: Math, dtype: int64
```

Exercise-1: Data Explore

1. Get dataset from online:

In [5]:
```
df = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iri
```

2. Print 1st 5 rows:

In [6]:
```
print(f"1st five rows: {df.head()}")
```

```
1st five rows:    sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

3. Print last 5 rows:

In [ ]:
```
print(f"last five rows: {df.tail()}")
```

```
last five rows:      sepal_length  sepal_width  petal_length  petal_width    species
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica
```

4. Print dataframe info:

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

5. Print Dataframe description

In [10]:
```
df.describe()
```

Out[10]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

6. Print the column of "species" & "sepal_length"

In [11]:
```
df[["species","sepal_length"]]
```

Out[11]:

	species	sepal_length
0	setosa	5.1
1	setosa	4.9
2	setosa	4.7
3	setosa	4.6
4	setosa	5.0
...	...	...
145	virginica	6.7
146	virginica	6.3
147	virginica	6.5
148	virginica	6.2
149	virginica	5.9

150 rows × 2 columns

7. Filter the rows in where "sepal_length" > 5 & "species" =='setosa' from the dataframe

In [12]:
```
df[(df["sepal_length"] > 5.0) & (df["species"] == 'setosa')]
```

Out[12]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
10	5.4	3.7	1.5	0.2	setosa
14	5.8	4.0	1.2	0.2	setosa
15	5.7	4.4	1.5	0.4	setosa
16	5.4	3.9	1.3	0.4	setosa
17	5.1	3.5	1.4	0.3	setosa
18	5.7	3.8	1.7	0.3	setosa
19	5.1	3.8	1.5	0.3	setosa
20	5.4	3.4	1.7	0.2	setosa
21	5.1	3.7	1.5	0.4	setosa
23	5.1	3.3	1.7	0.5	setosa
27	5.2	3.5	1.5	0.2	setosa
28	5.2	3.4	1.4	0.2	setosa
31	5.4	3.4	1.5	0.4	setosa
32	5.2	4.1	1.5	0.1	setosa
33	5.5	4.2	1.4	0.2	setosa
36	5.5	3.5	1.3	0.2	setosa
39	5.1	3.4	1.5	0.2	setosa
44	5.1	3.8	1.9	0.4	setosa
46	5.1	3.8	1.6	0.2	setosa
48	5.3	3.7	1.5	0.2	setosa

Handeling Missing Values using Pandas

Why need to handle missing values?

- ML models need clean and uninterrupted data in order to generate more accurate prediction on them.

Methods to handle missing values:

1. Dropping missing values
2. Fill missing values

3. Interpolation

Create a Dataset

In [174…
```
import pandas as pd
import numpy as np

# Creating a sample DataFrame with missing values
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Age': [25, np.nan, 30, 22, np.nan],
    'Salary': [50000, 54000, np.nan, 48000, 52000],
    'Department': ['HR', 'IT', 'IT', np.nan, 'Finance']
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

```
Original DataFrame:
      Name   Age   Salary Department
0    Alice  25.0  50000.0         HR
1      Bob   NaN  54000.0         IT
2  Charlie  30.0      NaN         IT
3    David  22.0  48000.0        NaN
4      Eve   NaN  52000.0    Finance
```

1. Drop Missing Data

1. a)Drop any row that contains at least one missing value:

In [137…
```
df_drop_any = df.dropna()
print(df_drop_any)
```

```
    Name   Age   Salary Department
0  Alice  25.0  50000.0         HR
```

1. b) Drop rows only if all values are missing:

In [138…
```
df_drop_all = df.dropna(how='all')
print(df_drop_all)
```

```
      Name   Age   Salary Department
0    Alice  25.0  50000.0         HR
1      Bob   NaN  54000.0         IT
2  Charlie  30.0      NaN         IT
3    David  22.0  48000.0        NaN
4      Eve   NaN  52000.0    Finance
```

1. c) Drop columns with missing values:

In [139…
```
df_drop_cols = df.dropna(axis=1)
print(df_drop_cols)
```

```
      Name
0    Alice
1      Bob
2  Charlie
3    David
4      Eve
```

2. Fill Missing Data

2. a) Fill with a constant:

In [141…
```
df_fill_const = df.fillna(0)
print(df_fill_const)
```

```
      Name   Age   Salary Department
0    Alice  25.0  50000.0         HR
1      Bob   0.0  54000.0         IT
2  Charlie  30.0      0.0         IT
3    David  22.0  48000.0          0
4      Eve   0.0  52000.0    Finance
```

2. b) Fill with forward fill (last valid value):

In [142…
```
df_fill_ffill = df.ffill()
print(df_fill_ffill)
```

```
      Name   Age   Salary Department
0    Alice  25.0  50000.0         HR
1      Bob  25.0  54000.0         IT
2  Charlie  30.0  54000.0         IT
3    David  22.0  48000.0         IT
4      Eve  22.0  52000.0    Finance
```

2. c) Fill with backward fill (next valid value):

In [143…
```
df_fill_bfill = df.bfill()
print(df_fill_bfill)
```

```
      Name   Age   Salary Department
0    Alice  25.0  50000.0         HR
1      Bob  30.0  54000.0         IT
2  Charlie  30.0  48000.0         IT
3    David  22.0  48000.0    Finance
4      Eve   NaN  52000.0    Finance
```

2. d) Fill each column with its mean/median:

In [146…
```
df_fill_mean = df.copy()

df_fill_mean['Age'] = df_fill_mean['Age'].fillna(df['Age'].mean())
df_fill_mean['Salary'] = df_fill_mean['Salary'].fillna(df['Salary'].mean())

print(df_fill_mean)
```

```
      Name        Age   Salary Department
0    Alice  25.000000  50000.0         HR
1      Bob  25.666667  54000.0         IT
2  Charlie  30.000000  51000.0         IT
3    David  22.000000  48000.0        NaN
4      Eve  25.666667  52000.0    Finance
```

3. Interpolation

In [147…
```
df_interpolated = df.copy()
df_interpolated[['Age','Salary']] = df_interpolated[['Age','Salary']].interpolate()

print(df_interpolated)
```

```
      Name   Age   Salary Department
0    Alice  25.0  50000.0         HR
1      Bob  27.5  54000.0         IT
2  Charlie  30.0  51000.0         IT
3    David  22.0  48000.0        NaN
4      Eve  22.0  52000.0    Finance
```

Data Transformation

- Renaming Columns
- Changing Data Types
- Creating or Modifying column

1. Renaming Column

In [148…
```
df.rename(columns={'Name':'Employee Name','Salary':'Monthly Salary'},inplace=True)

print(df)
```

```
  Employee Name   Age  Monthly Salary Department
0         Alice  25.0         50000.0         HR
1           Bob   NaN         54000.0         IT
2       Charlie  30.0             NaN         IT
3         David  22.0         48000.0        NaN
4           Eve   NaN         52000.0    Finance
```

2. Changing Data Types

In [152…
```
df['Age'] = df['Age'].fillna(df['Age'].mean())          # Fill missing before conver
df['Age'] = df['Age'].astype(int)                       # Convert to integer
```

```
df['Monthly Salary'] = df['Monthly Salary'].astype(float)  # Ensure float

# Convert Department to string and fill missing
df['Department'] = df['Department'].astype(str)
df['Department'] = df['Department'].ffill()


print(df)
```

```
  Employee Name  Age  Monthly Salary Department
0         Alice   25         50000.0         HR
1           Bob   25         54000.0         IT
2       Charlie   30             NaN         IT
3         David   22         48000.0        nan
4           Eve   25         52000.0    Finance
```

Exercise-2: Clean a Dataset by Handeling Missing Values and Renaming Columns

1. Random Dataset Creation

In [175…

```
import pandas as pd
import numpy as np

# Set seed for reproducibility
np.random.seed(54)

# Number of rows
n = 100

# Create a DataFrame with synthetic data
data = {
    'ID': range(1, n+1),
    'Name': np.random.choice(['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', '
    'Age': np.random.randint(18, 65, size=n).astype(float),  # Use float for missin
    'Salary': np.random.randint(30000, 100000, size=n).astype(float),
    'Department': np.random.choice(['HR', 'IT', 'Finance', 'Marketing', 'Operations
    'JoinDate': pd.date_range(start='2020-01-01', periods=n, freq='D')
}

df_big = pd.DataFrame(data)

# Introduce missing values randomly in a few columns (10% missing rate per column)
for col in ['Name', 'Age', 'Salary', 'Department']:
    df_big.loc[df_big.sample(frac=0.1).index, col] = np.nan

print(df_big.head(10))
```

```
   ID   Name   Age   Salary  Department   JoinDate
0   1  Frank  64.0      NaN     Finance 2020-01-01
1   2    NaN  51.0  99362.0          HR 2020-01-02
2   3    Bob  37.0  78731.0          HR 2020-01-03
3   4  Alice  26.0  86422.0  Operations 2020-01-04
4   5  Alice  47.0  42658.0          HR 2020-01-05
5   6    Bob  52.0  81439.0   Marketing 2020-01-06
6   7    Bob  62.0  48431.0     Finance 2020-01-07
7   8  Grace   NaN  31223.0          IT 2020-01-08
8   9    Bob  36.0  89917.0   Marketing 2020-01-09
9  10    Eve  40.0  55979.0          IT 2020-01-10
```

2. Explore Dataset

In [176…

```
#View the Structure:
df_big.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   ID          100 non-null    int64         
 1   Name        90 non-null     object        
 2   Age         90 non-null     float64       
 3   Salary      90 non-null     float64       
 4   Department  90 non-null     object        
 5   JoinDate    100 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(2), int64(1), object(2)
memory usage: 4.8+ KB
```

In [177…

```
#Summary Statistics:
df_big.describe(include='all')
```

Out[177…

	ID	Name	Age	Salary	Department	JoinDate
count	100.000000	90	90.000000	90.000000	90	100
unique	NaN	7	NaN	NaN	5	NaN
top	NaN	Eve	NaN	NaN	IT	NaN
freq	NaN	17	NaN	NaN	20	NaN
mean	50.500000	NaN	42.122222	67223.400000	NaN	2020-02-19 12:00:00
min	1.000000	NaN	18.000000	30766.000000	NaN	2020-01-01 00:00:00
25%	25.750000	NaN	32.000000	51366.250000	NaN	2020-01-25 18:00:00
50%	50.500000	NaN	42.000000	66653.000000	NaN	2020-02-19 12:00:00
75%	75.250000	NaN	53.000000	87340.750000	NaN	2020-03-15 06:00:00
max	100.000000	NaN	64.000000	99362.000000	NaN	2020-04-09 00:00:00
std	29.011492	NaN	13.428820	20196.998282	NaN	NaN

In [178…

```
#Check Missing Values:
df_big.isnull().sum()
```

Out[178…

```
ID             0
Name          10
Age           10
Salary        10
Department    10
JoinDate       0
dtype: int64
```

3. Handling Missing Data

In [179…

```
#Numeric Columns Example:
df_big[['Age', 'Salary']] = df_big[['Age', 'Salary']].interpolate()
df_big['Salary'] = df_big['Salary'].bfill()

print(df_big)
```

```
     ID   Name   Age   Salary  Department   JoinDate
0     1  Frank  64.0  99362.0     Finance 2020-01-01
1     2    NaN  51.0  99362.0          HR 2020-01-02
2     3    Bob  37.0  78731.0          HR 2020-01-03
3     4  Alice  26.0  86422.0  Operations 2020-01-04
4     5  Alice  47.0  42658.0          HR 2020-01-05
..  ...    ...   ...      ...         ...        ...
95   96  Alice  32.0  64324.0          IT 2020-04-05
96   97    Eve  37.0  79982.5          IT 2020-04-06
97   98    Bob  40.0  95641.0         NaN 2020-04-07
98   99  David  21.0  97589.0          IT 2020-04-08
99  100  Alice  44.0  98645.0   Marketing 2020-04-09

[100 rows x 6 columns]
```

In [180…

```
#Categorical Columns Example:
df_big['Name'] = df_big['Name'].ffill()  # Forward fill for names
df_big['Department'] = df_big['Department'].fillna(df_big['Department'].mode()[0])

print(df_big)
```

```
     ID   Name   Age   Salary  Department   JoinDate
0     1  Frank  64.0  99362.0     Finance 2020-01-01
1     2  Frank  51.0  99362.0          HR 2020-01-02
2     3    Bob  37.0  78731.0          HR 2020-01-03
3     4  Alice  26.0  86422.0  Operations 2020-01-04
4     5  Alice  47.0  42658.0          HR 2020-01-05
..  ...    ...   ...      ...         ...        ...
95   96  Alice  32.0  64324.0          IT 2020-04-05
96   97    Eve  37.0  79982.5          IT 2020-04-06
97   98    Bob  40.0  95641.0          IT 2020-04-07
98   99  David  21.0  97589.0          IT 2020-04-08
99  100  Alice  44.0  98645.0   Marketing 2020-04-09

[100 rows x 6 columns]
```

4. Transform Your Data

In [181…

```
# Renaming Columns:
df_big.rename(columns={'Name': 'Employee Name'}, inplace=True)
# Changing Data Types:
df_big['Age'] = df_big['Age'].astype(int)
df_big['Salary'] = df_big['Salary'].astype(float)


print(df_big)
```

```
     ID Employee Name  Age   Salary  Department   JoinDate
0     1         Frank   64  99362.0     Finance 2020-01-01
1     2         Frank   51  99362.0          HR 2020-01-02
2     3           Bob   37  78731.0          HR 2020-01-03
3     4         Alice   26  86422.0  Operations 2020-01-04
4     5         Alice   47  42658.0          HR 2020-01-05
..  ...           ...  ...      ...         ...        ...
95   96         Alice   32  64324.0          IT 2020-04-05
96   97           Eve   37  79982.5          IT 2020-04-06
97   98           Bob   40  95641.0          IT 2020-04-07
98   99         David   21  97589.0          IT 2020-04-08
99  100         Alice   44  98645.0   Marketing 2020-04-09

[100 rows x 6 columns]
```

In [182…
```
# Creating New Columns:
df_big['Bonus Salary'] = df_big['Salary'] * 1.10

print(df_big)
```

```
     ID Employee Name  Age   Salary  Department   JoinDate  Bonus Salary
0     1         Frank   64  99362.0     Finance 2020-01-01     109298.20
1     2         Frank   51  99362.0          HR 2020-01-02     109298.20
2     3           Bob   37  78731.0          HR 2020-01-03      86604.10
3     4         Alice   26  86422.0  Operations 2020-01-04      95064.20
4     5         Alice   47  42658.0          HR 2020-01-05      46923.80
..  ...           ...  ...      ...         ...        ...           ...
95   96         Alice   32  64324.0          IT 2020-04-05      70756.40
96   97           Eve   37  79982.5          IT 2020-04-06      87980.75
97   98           Bob   40  95641.0          IT 2020-04-07     105205.10
98   99         David   21  97589.0          IT 2020-04-08     107347.90
99  100         Alice   44  98645.0   Marketing 2020-04-09     108509.50

[100 rows x 7 columns]
```

In [186…
```
print(df_big.head(10))
```

```
   ID Employee Name  Age   Salary  Department   JoinDate  Bonus Salary
0   1         Frank   64  99362.0     Finance 2020-01-01      109298.2
1   2         Frank   51  99362.0          HR 2020-01-02      109298.2
2   3           Bob   37  78731.0          HR 2020-01-03       86604.1
3   4         Alice   26  86422.0  Operations 2020-01-04       95064.2
4   5         Alice   47  42658.0          HR 2020-01-05       46923.8
5   6           Bob   52  81439.0   Marketing 2020-01-06       89582.9
6   7           Bob   62  48431.0     Finance 2020-01-07       53274.1
7   8         Grace   49  31223.0          IT 2020-01-08       34345.3
8   9           Bob   36  89917.0   Marketing 2020-01-09       98908.7
9  10           Eve   40  55979.0          IT 2020-01-10       61576.9
```

Combining & Merging Dataframes

- Concatenation
- Merging
- Joining

1. Concatenation:

In [6]:
```
import pandas as pd

# Define DataFrames for concatenation
df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']
})
df2 = pd.DataFrame({
    'A': ['A3', 'A4', 'A5'],
    'B': ['B3', 'B4', 'B5']
})

# Concatenate the DataFrames vertically (row-wise)
concatenated_df = pd.concat([df1, df2],ignore_index=True)
print("Concatenated DataFrame:")
print(concatenated_df)
```

```
Concatenated DataFrame:
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3
4  A4  B4
5  A5  B5
```

2. Merging

In [7]:
```
import pandas as pd

# Define DataFrames for merging (with a common key)
df_left = pd.DataFrame({
    'key': ['K0', 'K1', 'K2', 'K3'],
    'A': ['A0', 'A1', 'A2', 'A3']
})
df_right = pd.DataFrame({
    'key': ['K0', 'K1', 'K2', 'K3'],
    'B': ['B0', 'B1', 'B2', 'B3']
})

# Merge the DataFrames on the 'key' column
merged_df = pd.merge(df_left, df_right, on='key')
print("Merged DataFrame:")
print(merged_df)
```

```
Merged DataFrame:
  key   A   B
0  K0  A0  B0
1  K1  A1  B1
2  K2  A2  B2
3  K3  A3  B3
```

3. Joining

In [8]:
```
import pandas as pd

# Define DataFrames for joining (using indexes)
df_left2 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2']
}, index=['K0', 'K1', 'K2'])
df_right2 = pd.DataFrame({
    'B': ['B0', 'B1', 'B2']
}, index=['K0', 'K1', 'K2'])

# Join the DataFrames on their indexes
joined_df = df_left2.join(df_right2)
print("Joined DataFrame:")
print(joined_df)
```

```
Joined DataFrame:
     A   B
K0  A0  B0
K1  A1  B1
K2  A2  B2
```

Grouping Data by Catagories

Why Group Data?

- Grouping data allows to perform operation on subset of data based on shared catagories.

groupby in Pandas:

- Operations:
  - Iterate over groups
  - Apply aggregation

Create Data

In [10]:
```
import pandas as pd

# Create a sample DataFrame
data = {
    'Team': ['A', 'B', 'A', 'B', 'A', 'B'],
    'Points': [10, 20, 30, 40, 50, 60],
    'Player': ['John', 'Alice', 'Bob', 'Carol', 'Dave', 'Eve']
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

```
Original DataFrame:
  Team  Points Player
0    A      10   John
1    B      20  Alice
2    A      30    Bob
3    B      40  Carol
4    A      50   Dave
5    B      60    Eve
```

Iterate over groups

In [12]:
```python
# Group the DataFrame by the 'Team' column
grouped = df.groupby('Team')

# Iterate over each group
for team, group in grouped:
    print(f"\nGroup: {team}")
    print(group)
```

```
Group: A
  Team  Points Player
0    A      10   John
2    A      30    Bob
4    A      50   Dave

Group: B
  Team  Points Player
1    B      20  Alice
3    B      40  Carol
5    B      60    Eve
```

Aggregation Functions:

- Using *groupby*
- Using *pivot_table*
- Custom Aggregation
  - Apply custom function using .agg()

Calculating Summary Statistics for Grouped Data

- Common Statistics:
  - Mean
  - Max
  - Min

Create Data

In [13]:
```python
import pandas as pd
import numpy as np

# Create a sample DataFrame
data = {
```

```
    'Team': ['A', 'B', 'A', 'B', 'A', 'B'],
    'Points': [10, 20, 30, 40, 50, 60],
    'Assists': [1, 2, 3, 4, 5, 6]
}

df = pd.DataFrame(data)
print("Original DataFrame:")
print(df)
```

```
Original DataFrame:
  Team  Points  Assists
0    A      10        1
1    B      20        2
2    A      30        3
3    B      40        4
4    A      50        5
5    B      60        6
```

1. Aggregation using *groupby*

In [14]:
```
#Calculate the sum of 'Points' and 'Assists' for each team
groupby_agg = df.groupby('Team').sum()
print("\nAggregation using groupby (sum):")
print(groupby_agg)
```

```
Aggregation using groupby (sum):
      Points  Assists
Team
A         90        9
B        120       12
```

2. Aggregation using pivot_table

In [16]:
```
pivot_agg = pd.pivot_table(df, index='Team', values='Points', aggfunc='mean')
print("\nAggregation using pivot_table (mean):")
print(pivot_agg)
```

```
Aggregation using pivot_table (mean):
      Points
Team
A       30.0
B       40.0
```

3. Custom Aggregation using .agg()

In [17]:
```
custom_agg = df.groupby('Team').agg({
    'Points': ['sum', 'mean', lambda x: np.max(x) - np.min(x)],
    'Assists': ['min', 'max']
})

# Optionally, rename columns for clarity
custom_agg.columns = ['Points_sum', 'Points_mean', 'Points_range', 'Assists_min', '
```

```
print("\nCustom Aggregation using .agg():")
print(custom_agg)
```

```
Custom Aggregation using .agg():
      Points_sum  Points_mean  Points_range  Assists_min  Assists_max
Team
A             90         30.0            40            1            5
B            120         40.0            40            2            6
```