Prepared by -: Vithusan

Pandas is a powerful Python library used for data analysis and manipulation.

Here's a quick overview of Pandas:

What it does:

- Loads and cleans datasets of various formats (CSV, Excel, SQL databases, etc.).
- Creates and manipulates data structures like DataFrames (similar to spreadsheets) and Series (single arrays).
- Performs data analysis tasks like filtering, sorting, grouping, aggregating, and statistical calculations.
- Enables data visualization through built-in plotting functions and integration with other libraries like Matplotlib.

Why it's popular:

- Easy to learn: User-friendly syntax and extensive documentation make it accessible to users of all levels.
- Powerful and versatile: Handles a wide range of data types and analysis tasks.
- Integrates well with other libraries: Works seamlessly with popular scientific computing libraries like NumPy and SciPy.
- Open-source and community-driven: Continuously improving with active development and a helpful community.

Who uses it:

- Data scientists, analysts, and researchers.
- Financial analysts and economists.
- Machine learning engineers and developers.
- Anyone who needs to work with and analyze data effectively.

What is Data Frames?

DataFrames are the **backbone of Pandas**, serving as the primary data structure for holding and manipulating data. Think of them as **flexible, multi-dimensional tables** similar to spreadsheets, but with much more power and functionality.

Here's a closer look at DataFrames:

Structure:

- **Rows:** Represent individual records or observations.
- **Columns:** Represent variables or features within each record.
- **Cells:** Intersection of rows and columns, containing specific data points.

Data Types:

- Can hold various data types in each cell, such as numbers, strings, booleans, dates, and even other DataFrames (nested!).
- Allows mixing data types within columns, providing flexibility for diverse data sets.

Key Features:

- **Indexing and selection:** Access specific rows, columns, or cells using labels, positions, or logical conditions.
- **Operations:** Perform calculations, aggregations, filtering, and sorting on data within columns or rows.
- **Merging and joining:** Combine data from multiple DataFrames based on shared information.
- **Visualization:** Easily visualize data patterns and relationships through built-in plotting functions.

Benefits:

- **Organized data representation:** Provides a clear and structured way to view and work with complex data sets.
- **Efficient data manipulation:** Offers powerful tools for cleaning, analyzing, and preparing data for further analysis.
- **Flexibility and versatility:** Adapts to various data types and analysis needs, making it a versatile tool for diverse tasks.

In summary, DataFrames are the workhorses of Pandas. They offer a user-friendly and powerful way to manage and analyze data, making them essential for anyone working with data science, analytics, or research.

In [149]:
```
#import the pandas

import pandas as pd
import numpy as np
```

In [150]:
```
#playing with dataframe

df = pd.DataFrame(np.arange(0, 24).reshape(6, 4), index = ["Row1", '
```

In [151]:
```
df.head()
```

Out[151]:

	Column1	Column2	Column3	Column4
Row1	0	1	2	3
Row2	4	5	6	7
Row3	8	9	10	11
Row4	12	13	14	15
Row5	16	17	18	19

NOTE ==> head(): This is a built-in method of pandas. DataFrame. It returns the first five rows of the DataFrame.

In [152]:
```
df.to_csv("test.csv")
```

to_csv() Built-in method in Pandas to save DataFrames as CSV files.

Accessing DataFrame Elements: Cheat Sheet

Label-based:

- **Single element:** `df['column']['row']`
- **Column:** `df['column_name']`
- **Multiple columns:** `df[['col1', 'col2']]`

Position-based (iloc):

- **Row:** `df.iloc[row_index]`
- **Specific element:** `df.iloc[row_index, col_index]`
- **Subset:** `df.iloc[start_row:end_row, start_col:end_col]`

Boolean Indexing:

- **Conditionally select rows:** `df[df['column'] > value]`

Tips:

- Indices start from 0.
- Use `df.head()` to understand structure.
- More advanced options in Pandas documentation.

Bonus:

- Use `df.loc` for non-integer labels.
- Use `df.at` / `df.iat` for faster scalar access.

In [153]:
```
df = pd.DataFrame(np.arange(0, 24).reshape(6, 4), index = ["Row1", '
```

In [154]:
```
df.head()
```

Out[154]:

	Column1	Column2	Column3	Column4
Row1	0	1	2	3
Row2	4	5	6	7
Row3	8	9	10	11
Row4	12	13	14	15
Row5	16	17	18	19

In [155]:
```
df["Column1"]["Row1"]
```

Out[155]:
```
0
```

In [156]:
```
df["Column1"]
```

Out[156]:
```
Row1     0
Row2     4
Row3     8
Row4    12
Row5    16
Row5    20
Name: Column1, dtype: int32
```

In [157]:
```
df[["Column1", "Column2"]]
```

Out[157]:

	Column1	Column2
Row1	0	1
Row2	4	5
Row3	8	9
Row4	12	13
Row5	16	17
Row5	20	21

In [158]:
```
df.loc["Row1"]
```

Out[158]:
```
Column1    0
Column2    1
Column3    2
Column4    3
Name: Row1, dtype: int32
```

In [159]:
```
type(df.loc["Row1"])
```

Out[159]:
```
pandas.core.series.Series
```

Data Series: Quick Guide

- **1. DataFrame Column:**
  - Single column of data within a Pandas DataFrame.
  - Think 1D list of specific variable/feature values.
  - Accessed & manipulated like DataFrames (focused on column).
- **2. Independent Data Sequence:**
  - Any ordered set of data points, not part of a DataFrame.
  - Temperatures over time, stock prices, etc.
  - Analyzed for trends, patterns, & relationships.

Remember:

- Consider context to determine meaning.
- Pandas: Data series = DataFrame column.

- Other contexts: Data series = any independent data sequence.
- Series can be either one row or one column.

In [160]:
```
df.iloc[0:2, 0:2]
```

Out[160]:

	Column1	Column2
Row1	0	1
Row2	4	5

In [161]:
```
type(df.iloc[0:2, 0:2])
```

Out[161]:
```
pandas.core.frame.DataFrame
```

In [162]:
```
type(df.iloc[0:1, 0])
```

Out[162]:
```
pandas.core.series.Series
```

DataFrame to Array: Short Guide

- **`to_numpy()`** : Entire DataFrame to NumPy array (copy).
- **`.values`** : Access underlying NumPy array directly.
- **Specific columns:** `df['col_name'].to_numpy()` .
- **Multiple columns:** `df.iloc[:, [0, 2]].to_numpy()` (precise selection).

Tips:

- Choose method based on your needs (entire/specific data).
- `to_numpy()` creates a copy, `.values` accesses directly.
- DataFrames can have mixed types, arrays typically single type.

In [163]:
```
df.iloc[:, 1:].values
```

Out[163]:
```
array([[ 1,  2,  3],
       [ 5,  6,  7],
       [ 9, 10, 11],
       [13, 14, 15],
       [17, 18, 19],
       [21, 22, 23]])
```

In [164]:
```
df.iloc[:, 1:].values.shape
```

Out[164]:
```
(6, 3)
```

In [165]:
```
df.isnull() #FIND THE NULL VALUES IN DATAFRAME
```

Out[165]:

	Column1	Column2	Column3	Column4
Row1	False	False	False	False
Row2	False	False	False	False
Row3	False	False	False	False
Row4	False	False	False	False
Row5	False	False	False	False
Row5	False	False	False	False

In [166]:
```
df.isnull().sum() #COUNT THE NULL VALUES
```

Out[166]:
```
Column1    0
Column2    0
Column3    0
Column4    0
dtype: int64
```

In [167]:
```
df["Column1"].value_counts() #COUNT THE UNIQUE VALUES
```

Out[167]:
```
Column1
0     1
4     1
8     1
12    1
16    1
20    1
Name: count, dtype: int64
```

In [168]:
```
df["Column1"].unique() #EXTRACTS THE UNIQUE, NON-DUPLICATED VALUES V
```

Out[168]:
```
array([ 0,  4,  8, 12, 16, 20])
```

Reading Files with Pandas

- **CSV:** `pd.read_csv("file.csv")` - Common tabular data format.
- **Excel:** `pd.read_excel("file.xlsx")` - Spreadsheet format.
- **JSON:** `pd.read_json("file.json")` - Data interchange format.
- **Text:** `pd.read_fwf()`, `pd.read_table()` - Simple text formats.
- **SQL:** `pd.read_sql("SELECT * FROM table", engine)` - Database interaction.
- **More:** Additional formats with specific libraries/functions.

Tips:

- Specify file path.
- Customize reading with optional parameters (header, sep, etc.).

In [169]:
```
df = pd.read_csv("student.csv") #load cvs file
```

In [170]:
```
df.head() #See the first five rows
```

Out[170]:

	id	name	class	mark	gender
0	1	Vithu	Four	75	male
1	2	Nila	Three	85	female
2	3	Arnold	Three	55	male
3	4	Krish	Four	60	female
4	5	John	Four	60	female

In [171]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35 entries, 0 to 34
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      35 non-null     int64 
 1   name    35 non-null     object
 2   class   35 non-null     object
 3   mark    35 non-null     int64 
 4   gender  35 non-null     object
dtypes: int64(2), object(3)
memory usage: 1.5+ KB
```

df.info()

- **Provides quick overview of DataFrame structure and content.**
- **Shows:**
  - Rows & columns count
  - Column names & data types
  - Memory usage
  - Non-null value counts
- **Useful for:**
  - Exploring data structure
  - Checking for missing values
  - Verifying data types
  - Optimizing memory usage

Think: Data snapshot for quick understanding and analysis!

In [172]:
```
df.describe()
```

Out[172]:

	id	mark
count	35.000000	35.000000
mean	18.000000	74.657143
std	10.246951	16.401117
min	1.000000	18.000000
25%	9.500000	62.500000
50%	18.000000	79.000000
75%	26.500000	88.000000
max	35.000000	96.000000

`df.describe()`:

Purpose:

`df.describe()` is a powerful tool in Pandas for generating **descriptive statistics** of your DataFrame's numeric columns. It provides a concise summary of the **central tendency, spread, and distribution** of your data, helping you gain insights into its characteristics.

Output:

The output of `df.describe()` depends on the data types within your DataFrame. For **numeric columns** it typically consists of:

- **Count:** The number of non-null values in the column.
- **Mean:** The average value of the non-null entries.
- **Standard deviation:** A measure of how spread out the data is around the mean.
- **Minimum and maximum:** The lowest and highest values found in the column.
- **Percentiles:** Values that split the data into equal proportions (e.g., 25th percentile divides the data into 25% lower and 75% higher values).

Benefits:

- **Quick data exploration:** Get a snapshot of the distribution of your numerical data without running complex calculations.
- **Outlier detection:** Identify potential outliers that deviate significantly from the main body of data.
- **Skewness assessment:** See if the data distribution is skewed towards one side (asymmetrical).
- **Central tendency and spread:** Understand the typical "middle" and range of your data points.

Additional tips:

- You can control the displayed percentiles and other statistics using optional arguments in `df.describe()`.
- Use `df.describe(include='all')` to include descriptive statistics for object columns (e.g., unique value counts).
- Remember, `df.describe()` only summarizes numeric data. For non-numeric columns, consider alternative analysis methods.

Think of `df.describe()` as your statistical cheat sheet for understanding the numeric heart of your DataFrame.

In [173]:
```
#Get the unique category counts
df["mark"].value_counts()
```

Out[173]:
```
mark
88    7
55    5
78    3
79    3
75    2
69    2
60    2
85    2
90    1
86    1
81    1
54    1
65    1
18    1
94    1
89    1
96    1
Name: count, dtype: int64
```

In [174]:
```
df[df["mark"] >= 75]
```

Out[174]:

	id	name	class	mark	gender
0	1	Vithu	Four	75	male
1	2	Nila	Three	85	female
6	7	My John Rob	Fifth	78	male
7	8	Asruid	Five	85	male
8	9	Tes Qry	Six	78	male
10	11	Ronald	Six	89	female
11	12	Recky	Six	94	female
12	13	Kty	Seven	88	female
13	14	Bigy	Seven	88	female
14	15	Tade Row	Four	88	male
15	16	Gimmy	Four	88	male
17	18	Honny	Five	75	male
22	23	Herod	Eight	79	male
23	24	Tiddy Now	Seven	78	male
24	25	Giff Tow	Seven	88	male
25	26	Crelea	Seven	79	male
26	27	Big Nose	Three	81	female
27	28	Rojj Base	Seven	86	female
29	30	Reppy Red	Six	79	female
30	31	Marry Toeey	Four	88	male
31	32	Binn Rott	Seven	90	female
32	33	Kenn Rein	Six	96	female
34	35	Rows Noump	Six	88	female

Read CSV

In [175]:
```
from io import StringIO, BytesIO
```

In [176]:
```
data = ("col1, col2, col3\n"
        "x, y, 1\n"
        "a, b, 2\n"
        "c, d, 3")
```

In [177]:
```
type(data)
```

Out[177]:
```
str
```

In [178]:
```
pd.read_csv(StringIO(data))
```

Out[178]:

	col1	col2	col3
0	x	y	1
1	a	b	2
2	c	d	3

In [179]:
```
#Read a specific columns
df = pd.read_csv(StringIO(data), usecols = ["col1"])
```

In [180]:
```
df
```

Out[180]:

	col1
0	x
1	a
2	c

In [181]:
```
data = ("a, b, c, d\n"
        "1, 2, 3, 4\n"
        "5, 6, 7, 8\n"
        "9, 10, 11, 12")
```

In [182]:
```
print(data)
```

```
a, b, c, d
1, 2, 3, 4
5, 6, 7, 8
9, 10, 11, 12
```

In [183]:
```
# Read CSV data from a string, treating all columns as string object
df = pd.read_csv(StringIO(data), dtype = object)
```

In [184]:
```
df
```

Out[184]:

	a	b	c	d
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

In [185]:
```
df["a"][0]
```

Out[185]:
```
'1'
```

In [186]:
```
type(df["a"][0])
```

Out[186]:
```
str
```

In [187]:
```
df["a"]
```

Out[187]:
```
0    1
1    5
2    9
Name: a, dtype: object
```

In [188]:
```
df = pd.read_csv(StringIO(data), dtype = int)
```

In [189]:
```
type(df["a"][0])
```

Out[189]:
```
numpy.int32
```

In [190]:
```
df = pd.read_csv(StringIO(data), dtype = float)
```

In [191]:
```
type(df["a"][0])
```

Out[191]:
```
numpy.float64
```

In [192]:
```
#Give different datatype to each columns
df = pd.read_csv(StringIO(data), dtype={"a": "int64", "b": int, "c":
```

In [193]:
```
df
```

Out[193]:

	a	b	c	d
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

In [194]:
```
type(df["a"][1])
```

Out[194]:
```
numpy.int64
```

In [195]:
```
df.dtypes
```

Out[195]:
```
a      int64
 b     int64
 c     int64
 d     int64
dtype: object
```

In [196]:
```
data = ("index,a, b, c\n"
        "4, apple, bat, 5.7\n"
        "8, orange, cow, 10\n")
```

In [197]:
```
pd.read_csv(StringIO(data))
```

Out[197]:

	index	a	b	c
0	4	apple	bat	5.7
1	8	orange	cow	10.0

In [198]:
```
pd.read_csv(StringIO(data), index_col = 0)
```

Out[198]:

	a	b	c
index			
4	apple	bat	5.7
8	orange	cow	10.0

In [199]:
```
data = ("a, b, c\n"
        "4, apple, bat\n"
        "8, orange, cow\n")
```

In [200]:
```
pd.read_csv(StringIO(data))
```

Out[200]:

	a	b	c
0	4	apple	bat
1	8	orange	cow

In [201]:
```
#Combining usecols and index_col
pd.read_csv(StringIO(data), usecols = ["a"], index_col = False)
```

Out[201]:

	a
0	4
1	8

In [202]:
```
#Quoting and Escape Chracters, Very useful in NLP
data = 'a, b\n"hello, \\"Vithu\\", nice to see you", 5'
```

In [203]:
```
pd.read_csv(StringIO(data), escapechar = "\\")
```

Out[203]:

	a	b
0	hello, "Vithu", nice to see you	5

In [204]:
```
#URL TO CSV
df = pd.read_csv("https://raw.githubusercontent.com/thasvithu/Data-S
```

In [205]:
```
df.head()
```

Out[205]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [206]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object 
 2   color    53940 non-null  object 
 3   clarity  53940 non-null  object 
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64  
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

JSON to CSV

In [207]:
```
data = '{"id": 1, "name": "Thas Vithu", "position": "Data Scientist'

# Read JSON string into a Pandas DataFrame
df1 = pd.read_json(data, orient='index')

# Display the DataFrame
print(df1)
```

```
                          0
id                        1
name              Thas Vithu
position     Data Scientist
department     Data Science
```

In [208]:
```
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-da
```

In [209]:
```
df.head()
```

Out[209]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735

In [210]:
```
# Convert JSON into CSV
df.to_csv("wine.csv")
```

In [211]:
```
df1.to_json()
```

Out[211]:
```
'{"0":{"id":1,"name":"Thas Vithu","position":"Data Scientist","dep
artment":"Data Science"}}'
```

In [212]:
```
#df.to_json()
```

In [213]:
```
df.to_json(orient = "records")
```


```
0},{"0":1,"1":13.56,"2":1.71,"3":2.31,"4":16.2,"5":117,"6":3.1
5,"7":3.29,"8":0.34,"9":2.34,"10":6.13,"11":0.95,"12":3.38,"13":
795},{"0":1,"1":13.41,"2":3.84,"3":2.12,"4":18.8,"5":90,"6":2.4
5,"7":2.68,"8":0.27,"9":1.48,"10":4.28,"11":0.91,"12":3.0,"13":1
035},{"0":1,"1":13.88,"2":1.89,"3":2.59,"4":15.0,"5":101,"6":3.2
5,"7":3.56,"8":0.17,"9":1.7,"10":5.43,"11":0.88,"12":3.56,"13":1
095},{"0":1,"1":13.24,"2":3.98,"3":2.29,"4":17.5,"5":103,"6":2.6
4,"7":2.63,"8":0.32,"9":1.66,"10":4.36,"11":0.82,"12":3.0,"13":6
80},{"0":1,"1":13.05,"2":1.77,"3":2.1,"4":17.0,"5":107,"6":3.
0,"7":3.0,"8":0.28,"9":2.03,"10":5.04,"11":0.88,"12":3.35,"13":8
85},{"0":1,"1":14.21,"2":4.04,"3":2.44,"4":18.9,"5":111,"6":2.8
5,"7":2.65,"8":0.3,"9":1.25,"10":5.24,"11":0.87,"12":3.33,"13":1
080},{"0":1,"1":14.38,"2":3.59,"3":2.28,"4":16.0,"5":102,"6":3.2
5,"7":3.17,"8":0.27,"9":2.19,"10":4.9,"11":1.04,"12":3.44,"13":1
065},{"0":1,"1":13.9,"2":1.68,"3":2.12,"4":16.0,"5":101,"6":3.
1,"7":3.39,"8":0.21,"9":2.14,"10":6.1,"11":0.91,"12":3.33,"13":9
85},{"0":1,"1":14.1,"2":2.02,"3":2.4,"4":18.8,"5":103,"6":2.7
5,"7":2.92,"8":0.32,"9":2.38,"10":6.2,"11":1.07,"12":2.75,"13":1
060},{"0":1,"1":13.94,"2":1.73,"3":2.27,"4":17.4,"5":108,"6":2.8
```


Reading HTML Content

In [214]:
```
url = "https://www.fdic.gov/resources/resolutions/bank-failures/fail

dfs = pd.read_html(url)
```

In [215]:
```
dfs[0]
```

Out[215]:

	Bank NameBank	CityCity	StateSt	CertCert	Acquiring InstitutionAI	Closing DateClosing	FundFunc
0	Citizens Bank	Sac City	IA	8758	Iowa Trust & Savings Bank	November 3, 2023	10545
1	Heartland Tri-State Bank	Elkhart	KS	25851	Dream First Bank, N.A.	July 28, 2023	10544
2	First Republic Bank	San Francisco	CA	59017	JPMorgan Chase Bank, N.A.	May 1, 2023	10543
3	Signature Bank	New York	NY	57053	Flagstar Bank, N.A.	March 12, 2023	10540
4	Silicon Valley Bank	Santa Clara	CA	24735	First–Citizens Bank & Trust Company	March 10, 2023	10539
...	...	...	...	...	...	...	..
563	Superior Bank, FSB	Hinsdale	IL	32646	Superior Federal, FSB	July 27, 2001	6004
564	Malta National Bank	Malta	OH	6629	North Valley Bank	May 3, 2001	4648
565	First Alliance Bank & Trust Co.	Manchester	NH	34264	Southern New Hampshire Bank & Trust	February 2, 2001	4647
566	National State Bank of Metropolis	Metropolis	IL	3815	Banterra Bank of Marion	December 14, 2000	4646
567	Bank of Honolulu	Honolulu	HI	21029	Bank of the Orient	October 13, 2000	4645

568 rows × 7 columns

In [216]:
```
type(dfs)
```

Out[216]:
```
list
```

In [217]:
```
url_mcc = "https://en.wikipedia.org/wiki/Mobile_country_code"
dfs = pd.read_html(url_mcc, match = "country", header = 0)
```

In [218]:
```
dfs[0]
```

Out[218]:

	Mobile country code	Country	ISO 3166	Mobile network codes	National MNC authority	Remarks
0	289	A Abkhazia	GE-AB	List of mobile network codes in Abkhazia	NaN	MCC is not listed by ITU
1	412	Afghanistan	AF	List of mobile network codes in Afghanistan	NaN	NaN
2	276	Albania	AL	List of mobile network codes in Albania	NaN	NaN
3	603	Algeria	DZ	List of mobile network codes in Algeria	NaN	NaN
4	544	American Samoa (United States of America)	AS	List of mobile network codes in American Samoa	NaN	NaN
...	...	...	...	...	...	...
247	452	Vietnam	VN	List of mobile network codes in the Vietnam	NaN	NaN
248	543	W Wallis and Futuna	WF	List of mobile network codes in Wallis and Futuna	NaN	NaN
249	421	Y Yemen	YE	List of mobile network codes in the Yemen	NaN	NaN
250	645	Z Zambia	ZM	List of mobile network codes in Zambia	NaN	NaN
251	648	Zimbabwe	ZW	List of mobile network codes in Zimbabwe	NaN	NaN

252 rows × 6 columns

Reading Excel Files

In [219]:
```
df_excel = pd.read_excel("ds_salaries.xlsx")
```

In [220]:
```
type(df_excel)
```

Out[220]:
```
pandas.core.frame.DataFrame
```

In [221]:
```
df_excel.head()
```

Out[221]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency
0	2023	SE	FT	Principal Data Scientist	80000	EUR
1	2023	MI	CT	ML Engineer	30000	USD
2	2023	MI	CT	ML Engineer	25500	USD
3	2023	SE	FT	Data Scientist	175000	USD
4	2023	SE	FT	Data Scientist	120000	USD

Pickling

- All pandas objects are equipped with to_pickle methods which use Pyton's cPickle module to save data structures to disk using the pickle format.

In [222]:
```
df_excel.to_pickle("pickleFile.xlsx")
```

In [223]:
```
df = pd.read_pickle("pickleFile.xlsx")
```

In [224]:
```
df.head()
```

Out[224]:

	work_year	experience_level	employment_type	job_title	salary	salary_currency
0	2023	SE	FT	Principal Data Scientist	80000	EUR
1	2023	MI	CT	ML Engineer	30000	USD
2	2023	MI	CT	ML Engineer	25500	USD
3	2023	SE	FT	Data Scientist	175000	USD
4	2023	SE	FT	Data Scientist	120000	USD

2024.01.14

[Find Me on Linkedin (https://bit.ly/3U2fXs6)]

[Github Pandas Repository (https://bit.ly/3vsbd4L)]