SANJANA DEVI

AI/ML Student | Intern @ CodroidHub Private Limited | First-Year Engineering Student.

Artificial Intelligence / Machine Learning Bootcamp

Python Library : pandas

What is Pandas?

Pandas is an open-source Python library used for data manipulation, analysis, and cleaning. It allows you to work with data in table form (like Excel) using DataFrames and Series.

Why Use Pandas?

Easy to load and read files like CSV, Excel, JSON

Helps in cleaning, filtering, sorting data

Makes data analysis faster and simpler

Works well with other libraries like NumPy and Matplotlib

Creating Data

In [61…

```
import pandas as pd # Data Analysis
```

In [62…

```
print(dir("pandas"))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_
subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__'
, '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook_
_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_
map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', '
removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split'
, 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Function	Description
`pd.Series()`	Create a one-dimensional labeled array
`pd.DataFrame()`	Create a 2D table (rows and columns)
`pd.read_csv()`	Read data from a CSV file
`pd.read_excel()`	Read data from an Excel file
`pd.read_json()`	Read JSON data
`pd.DataFrame.from_dict()`	Create DataFrame from dictionary

In [63…

```
df1 = pd.read_csv('final_dataset.csv') #Dataframe
```

In [64…

```
df=pd.DataFrame(df1)
print(df)
```

```
      Date  Month  Year  Holidays_Count  Days   PM2.5    PM10     NO2    SO2  \
0        1      1  2021               0     5  408.80  442.42  160.61  12.95   
1        2      1  2021               0     6  404.04  561.95   52.85   5.18   
2        3      1  2021               1     7  225.07  239.04  170.95  10.93   
3        4      1  2021               0     1   89.55  132.08  153.98  10.42   
4        5      1  2021               0     2   54.06   55.54  122.66   9.70   
...    ...    ...   ...             ...   ...     ...     ...     ...    ...   
1456    27     12  2024               0     5   58.43  249.17   41.69  65.89   
1457    28     12  2024               0     6   33.83  150.77   33.31  66.14   
1458    29     12  2024               1     7   31.21  139.75   27.01  65.94   
1459    30     12  2024               0     1   38.01  152.83   29.12  65.16   
1460    31     12  2024               0     2   80.42  318.96   40.37  64.98   

        CO  Ozone  AQI  
0     2.77  43.19  462  
1     2.60  16.43  482  
2     1.40  44.29  263  
3     1.01  49.19  207  
4     0.64  48.88  149  
...    ...    ...  ...  
1456  0.99  36.25  263  
1457  0.79  35.19  113  
1458  0.57  35.88  142  
1459  0.55  38.38  116  
1460  0.84  39.93  209  

[1461 rows x 12 columns]
```

2. Inspecting Data

In [65…
```
df.head()
```

Out[65]:

	Date	Month	Year	Holidays_Count	Days	PM2.5	PM10	NO2	SO2	CO	Ozone	AQI
0	1	1	2021	0	5	408.80	442.42	160.61	12.95	2.77	43.19	462
1	2	1	2021	0	6	404.04	561.95	52.85	5.18	2.60	16.43	482
2	3	1	2021	1	7	225.07	239.04	170.95	10.93	1.40	44.29	263
3	4	1	2021	0	1	89.55	132.08	153.98	10.42	1.01	49.19	207
4	5	1	2021	0	2	54.06	55.54	122.66	9.70	0.64	48.88	149

In [66…
```
df.tail()
```

Out[66]:

	Date	Month	Year	Holidays_Count	Days	PM2.5	PM10	NO2	SO2	CO	Ozone	AQI
1456	27	12	2024	0	5	58.43	249.17	41.69	65.89	0.99	36.25	263
1457	28	12	2024	0	6	33.83	150.77	33.31	66.14	0.79	35.19	113
1458	29	12	2024	1	7	31.21	139.75	27.01	65.94	0.57	35.88	142
1459	30	12	2024	0	1	38.01	152.83	29.12	65.16	0.55	38.38	116
1460	31	12	2024	0	2	80.42	318.96	40.37	64.98	0.84	39.93	209

In [67…
```
df.shape
```

Out[67]:
```
(1461, 12)
```

In [68…
```
df.columns
```

Out[68]:
```
Index(['Date', 'Month', 'Year', 'Holidays_Count', 'Days', 'PM2.5', 'PM10',
       'NO2', 'SO2', 'CO', 'Ozone', 'AQI'],
      dtype='object')
```

In [69…
```
df.index
```

Out[69]:
```
RangeIndex(start=0, stop=1461, step=1)
```

In [70…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1461 entries, 0 to 1460
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Date            1461 non-null   int64  
 1   Month           1461 non-null   int64  
 2   Year            1461 non-null   int64  
 3   Holidays_Count  1461 non-null   int64  
 4   Days            1461 non-null   int64  
 5   PM2.5           1461 non-null   float64
 6   PM10            1461 non-null   float64
 7   NO2             1461 non-null   float64
 8   SO2             1461 non-null   float64
 9   CO              1461 non-null   float64
 10  Ozone           1461 non-null   float64
 11  AQI             1461 non-null   int64  
dtypes: float64(6), int64(6)
memory usage: 137.1 KB
```

In [71…

```
df.describe()
```

Out[71]:

	Date	Month	Year	Holidays_Count	Days	PM2.5	PM10	NO2	SO2
count	1461.000000	1461.000000	1461.000000	1461.000000	1461.000000	1461.000000	1461.000000	1461.000000	1461.000000
mean	15.729637	6.522930	2022.501027	0.189596	4.000684	90.774538	218.219261	37.184921	20.104921
std	8.803105	3.449884	1.118723	0.392116	2.001883	71.650579	129.297734	35.225327	16.543659
min	1.000000	1.000000	2021.000000	0.000000	1.000000	0.050000	9.690000	2.160000	1.210000
25%	8.000000	4.000000	2022.000000	0.000000	2.000000	41.280000	115.110000	17.280000	7.710000
50%	16.000000	7.000000	2023.000000	0.000000	4.000000	72.060000	199.800000	30.490000	15.430000
75%	23.000000	10.000000	2024.000000	0.000000	6.000000	118.500000	297.750000	45.010000	26.620000
max	31.000000	12.000000	2024.000000	1.000000	7.000000	1000.000000	1000.000000	433.980000	113.400000

In [72…

```
df.dtypes
```

Out[72]:

```
Date                int64
Month               int64
Year                int64
Holidays_Count      int64
Days                int64
PM2.5             float64
PM10              float64
NO2               float64
SO2               float64
CO                float64
Ozone             float64
AQI                 int64
dtype: object
```

Note: What do `int64` and `float64` mean in pandas?

These are **data types** used by **NumPy** and **pandas** to represent numbers in a DataFrame or Series.

`int64` → **Integer** numbers (whole numbers)

- Example: `1`, `25`, `-7`, `1000`
- `int` = integer
- `64` = uses 64 bits of memory to store the value

It can store **very large or very small** whole numbers (positive or negative).

`float64` → **Floating point numbers** (decimal numbers)

- Example: `1.5`, `3.14`, `-7.0`, `2.0`
- `float` = decimal number
- `64` = uses 64 bits of memory to store the value

Used when the number has **decimal points**.

In [83…

```
import pandas as pd
```

```
data = {
    'Age': [23, 25, 22],
    'Marks': [85.5, 90.0, 78.25]
}

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

 #**Output:**


#Age       int64
#Marks    float64
#dtype: object


 #So:

#* `Age` column contains integers → `int64`
#* `Marks` column contains decimals → `float64`
```

```
Age        int64
Marks    float64
dtype: object
```

Function	Description
`df.head(n)`	First `n` rows
`df.tail(n)`	Last `n` rows
`df.shape`	Rows and columns (tuple)
`df.columns`	List of column names
`df.index`	List of row labels
`df.info()`	Summary of DataFrame
`df.describe()`	Stats of numeric columns
`df.dtypes`	Data types of columns

3. Selecting Data

In [84…

```
df["Age"]
```

Out[84]:

```
0    23
1    25
2    22
Name: Age, dtype: int64
```

In [85…

```
df[['Age', 'Marks']]
```

Out[85]:

	Age	Marks
0	23	85.50
1	25	90.00
2	22	78.25

In [86…

```
df.iloc[1]
```

Out[86]:

```
Age      25.0
Marks    90.0
Name: 1, dtype: float64
```

In [88…

```
df[df['Age'] > 9]
```

Out[88]:

	Age	Marks
0	23	85.50
1	25	90.00
2	22	78.25

Function	Description
`df['column']`	Select one column
`df[['col1', 'col2']]`	Select multiple columns

`df.iloc[row_idx]`	Select by index (position)
`df.loc[row_label]`	Select by label
`df[df['col'] > value]`	Conditional filter

4. Modifying Data

In [92…

```
df['name'] =["sa","df", "nj"]
print(df)
```

```
   Age  Marks Student_Name name
0   23  85.50           sa   sa
1   25  90.00           df   df
2   22  78.25           nj   nj
```

In [93…

```
df.rename(columns={'name': 'Student_Name'}, inplace=True)
print(df)
```

```
   Age  Marks Student_Name Student_Name
0   23  85.50           sa           sa
1   25  90.00           df           df
2   22  78.25           nj           nj
```

In [ ]:

```
df.drop("Student_Name", axis=3, inplace=True)
```

4. Modifying Data

Function	Description
`df['new_col'] = ...`	Create new column
`df.rename()`	Rename columns or index
`df.drop()`	Remove columns or rows
`df.insert()`	Insert new column at position
`df.replace()`	Replace values
`df.astype()`	Change data type
`df.fillna()`	Fill missing values
`df.dropna()`	Drop missing values

In [50…

```
df['name'] =["sa","df", "nj"]
```

In [51…

```
df
```

Out[51]:

	Age	Marks	name
0	23	85.50	sa
1	25	90.00	df
2	22	78.25	nj

5. Aggregation & Grouping

Function	Description
`df.sum()`	Sum
`df.mean()`	Average
`df.min()`, `df.max()`	Min and max
`df.count()`	Count non-NA values
`df.value_counts()`	Count of unique values
`df.groupby()`	Group and aggregate
`df.agg()`	Apply multiple aggregation

---

6. Sorting & Reordering

Function	Description
`df.sort_values()`	Sort by values
`df.sort_index()`	Sort by index
`df.reset_index()`	Reset index to default
`df.set_index()`	Set a column as index

7. Merging & Joining

Function	Description
`pd.concat()`	Combine along rows or columns
`pd.merge()`	Merge two DataFrames
`df.join()`	Join on index

8. Time Series Functions

Function	Description
`pd.to_datetime()`	Convert to datetime
`df.resample()`	Resample time-series data
`df.dt`	Access datetime attributes

9. File Operations

Function	Description
`df.to_csv()`	Export to CSV
`df.to_excel()`	Export to Excel
`df.to_json()`	Export to JSON

10. Other Useful Functions

Function	Description
`df.apply()`	Apply function to rows/columns
`df.map()`	Map values (for Series)
`df.isnull()`	Detect missing values
`df.notnull()`	Opposite of isnull
`df.duplicated()`	Find duplicates
`df.drop_duplicates()`	Remove duplicates

Connect @
Mail (Sanjana): sanjanadevibihana@gmail.com
Contact: +91-6283762268

SANJANA DEVI

AI/ML Student | Intern @ CodroidHub Private Limited | First-Year Engineering Student.

SANJANA DEVI