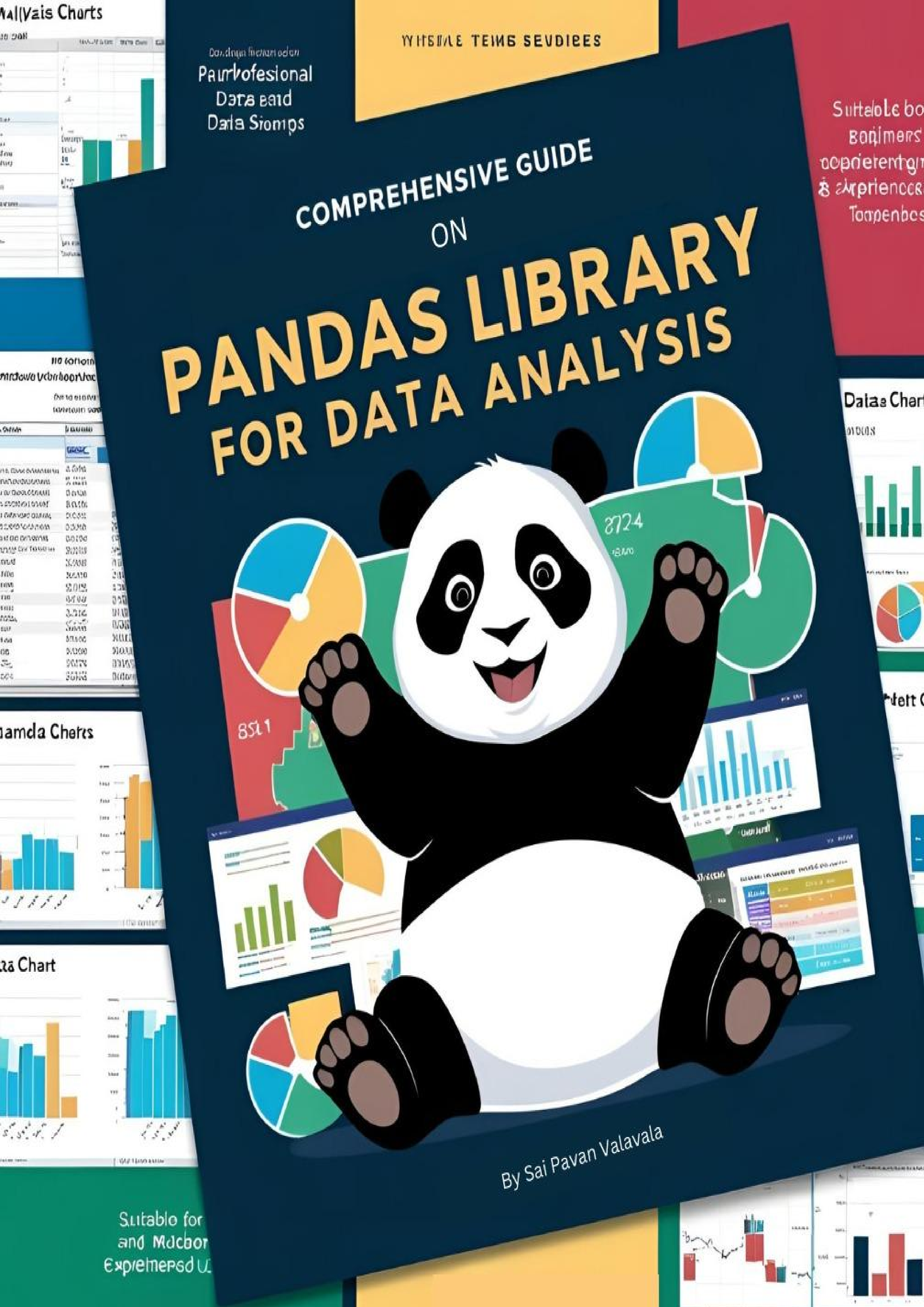
COMPREHENSIVE GUIDE
ON
PANDAS LIBRARY
FOR DATA ANALYSIS
By Sai Pavan Valavala

1. DataFrame Creation

- pd.DataFrame(): Create a DataFrame from lists, dictionaries, or NumPy arrays.
  Example: df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
- pd.Series(): Create a Series from lists, dictionaries, or NumPy arrays.
  Example: s = pd.Series([1, 2, 3, 4])
- pd.concat(): Concatenate DataFrames along a particular axis.
  Example: df_combined = pd.concat([df1, df2], axis=0)
- pd.merge(): Merge DataFrames using SQL-style join operations.
  Example: df_merged = pd.merge(df1, df2, on='key')
- pd.read_csv(): Read a CSV file into a DataFrame.
  Example: df = pd.read_csv('data.csv')
- pd.read_excel(): Read an Excel file into a DataFrame.
  Example: df = pd.read_excel('data.xlsx')
- pd.read_sql(): Read data from a SQL query into a DataFrame.
  Example: df = pd.read_sql('SELECT * FROM table', con)
- pd.read_json(): Read a JSON file into a DataFrame.
  Example: df = pd.read_json('data.json')
- pd.read_parquet(): Read data from a Parquet file into a DataFrame.
  Example: df = pd.read_parquet('data.parquet')
- pd.read_html(): Read data from an HTML table into a DataFrame.
  Example: df = pd.read_html('data.html')[0]

- pd.read_pickle(): Read data from a pickle file into a DataFrame.
  Example: df = pd.read_pickle('data.pkl')

---

2. Data Inspection

- df.head(): Display the first n rows of the DataFrame.
  Example: df.head()
- df.tail(): Display the last n rows of the DataFrame.
  Example: df.tail()
- df.shape: Get the dimensions (rows, columns) of the DataFrame.
  Example: df.shape
- df.info(): Display a summary of the DataFrame, including column types and counts of non-null values.
  Example: df.info()
- df.describe(): Get summary statistics for numerical columns.
  Example: df.describe()
- df.columns: Get the column names of the DataFrame.
  Example: df.columns
- df.dtypes: Get the data types of the DataFrame columns.
  Example: df.dtypes
- df.isnull(): Check for missing values (NaN).
  Example: df.isnull()
- df.notnull(): Check for non-missing values.
  Example: df.notnull()

- df.memory_usage(): Get the memory usage of each column.
  Example: df.memory_usage(deep=True)

---

3. Data Selection and Indexing

- df['column_name']: Access a specific column.
  Example: df['A']
- df.iloc[]: Indexing by position (integer-based).
  Example: df.iloc[0, 1]
- df.loc[]: Indexing by label (label-based).
  Example: df.loc[0, 'A']
- df.at[]: Access a single value by row/column label.
  Example: df.at[0, 'A']
- df.iat[]: Access a single value by row/column position.
  Example: df.iat[0, 1]
- df.query(): Query the DataFrame using a string expression.
  Example: df.query('A > 1')
- df.xs(): Get a cross-section from the DataFrame.
  Example: df.xs(0)
- df.set_index(): Set one or more columns as the index.
  Example: df.set_index('A', inplace=True)
- df.reset_index(): Reset the index to the default integer-based index.
  Example: df.reset_index()
- df.sort_index(): Sort the DataFrame by index labels.
  Example: df.sort_index()

4. Data Cleaning

- df.dropna(): Remove missing values (rows or columns).
  Example: df.dropna()
- df.fillna(): Fill missing values with a specified value or method.
  Example: df.fillna(0)
- df.replace(): Replace specified values with other values.
  Example: df.replace(0, np.nan)
- df.drop(): Drop specified rows or columns.
  Example: df.drop('A', axis=1, inplace=True)
- df.rename(): Rename columns or index labels.
  Example: df.rename(columns={'A': 'NewA'}, inplace=True)
- df.astype(): Change the data type of one or more columns.
  Example: df['A'] = df['A'].astype(float)
- df.duplicated(): Identify duplicate rows.
  Example: df.duplicated()
- df.drop_duplicates(): Remove duplicate rows.
  Example: df.drop_duplicates()
- df.isin(): Check if values in a column are present in another collection.
  Example: df['A'].isin([1, 2])
- df.str.*: String methods for text manipulation (e.g., df['column'].str.lower()).
  Example: df['A'] = df['A'].str.lower()

5. Aggregation and Grouping

- df.groupby(): Group the DataFrame by one or more columns.
  Example: df.groupby('A').sum()
- df.agg(): Apply one or more aggregation functions to grouped data.
  Example: df.groupby('A').agg({'B': 'sum'})
- df.pivot_table(): Create a pivot table to summarize data.
  Example: df.pivot_table(values='B', index='A', aggfunc='sum')
- df.crosstab(): Compute a cross-tabulation of two or more factors.
  Example: pd.crosstab(df['A'], df['B'])
- df.mean(), df.median(), df.mode(): Compute mean, median, and mode.
  Example: df['A'].mean()
- df.sum(), df.min(), df.max(), df.std(), df.var(): Calculate basic statistics.
  Example: df['A'].sum()
- df.count(): Count non-null values in a column or row.
  Example: df['A'].count()
- df.first(), df.last(): Get the first or last value from each group.
  Example: df.groupby('A').first()

6. Merging, Joining, and Concatenating

- df.merge(): Merge DataFrames using SQL-style joins.
  Example: df_merged = pd.merge(df1, df2, on='key')
- df.join(): Join another DataFrame using index or a column.
  Example: df1.join(df2, on='key')
- df.append(): Append rows to the DataFrame (deprecated).
  Example: df = df.append(df2)
- pd.concat(): Concatenate DataFrames along a specified axis.
  Example: df_combined = pd.concat([df1, df2], axis=0)
- df.update(): Update the DataFrame with values from another DataFrame.
  Example: df.update(df2)

---

7. Sorting and Ranking

- df.sort_values(): Sort the DataFrame by one or more columns.
  Example: df.sort_values(by='A')
- df.sort_index(): Sort by index labels.
  Example: df.sort_index()
- df.rank(): Rank the values in the DataFrame.
  Example: df['A'].rank()

- df.argsort(): Get the indices that would sort a column or series.
  Example: df['A'].argsort()

---

8. Reshaping and Pivoting

- df.melt(): Unpivot a DataFrame from wide to long format.
  Example: df_melted = df.melt(id_vars=['A'], value_vars=['B', 'C'])
- df.pivot(): Pivot a DataFrame from long to wide format.
  Example: df_pivoted = df.pivot(index='A', columns='B', values='C')
- df.pivot_table(): Create a pivot table to summarize data.
  Example: df.pivot_table(values='C', index='A', columns='B', aggfunc='sum')
- df.stack(): Stack columns into rows (MultiIndex).
  Example: df_stack = df.stack()
- df.unstack(): Unstack a level of the MultiIndex into columns.
  Example: df_unstack = df.unstack()
- df.transpose(): Transpose the DataFrame (rows become columns and vice versa).
  Example: df.T

---

9. Time Series Operations

- pd.to_datetime(): Convert a column to datetime format.
  Example: df['date'] = pd.to_datetime(df['date'])
- df['column'].dt.year: Extract the year from a datetime column.
  Example: df['year'] = df['date'].dt.year
- df['column'].dt.month: Extract the month from a datetime column.
  Example: df['month'] = df['date'].dt.month
- df['column'].dt.day: Extract the day from a datetime column.
  Example: df['day'] = df['date'].dt.day
- df['column'].dt.weekday: Get the weekday (0 = Monday, 6 = Sunday).
  Example: df['weekday'] = df['date'].dt.weekday
- df['column'].dt.daysinmonth: Get the number of days in a month.
  Example: df['days_in_month'] = df['date'].dt.daysinmonth
- df['column'].dt.is_month_end: Check if the date is the end of the month.
  Example: df['is_month_end'] = df['date'].dt.is_month_end
- df['column'].dt.is_month_start: Check if the date is the start of the month.
  Example: df['is_month_start'] = df['date'].dt.is_month_start
- df.resample(): Resample time-series data at a different frequency.
  Example: df_resampled = df.resample('D').mean()

- df.shift(): Shift data values for time-series (useful for calculating differences).
  Example: df['shifted'] = df['A'].shift(1)
- df.rolling(): Apply rolling window functions (e.g., moving averages).
  Example: df['rolling_avg'] = df['A'].rolling(window=3).mean()
- df.expanding(): Apply expanding window functions.
  Example: df['expanding_sum'] = df['A'].expanding().sum()
- df.ewm(): Apply exponentially weighted functions for time-series.
  Example: df['ewm'] = df['A'].ewm(span=10).mean()

---

10. String Operations

- df.str.contains(): Check if a substring is in a string column.
  Example: df['A'].str.contains('text')
- df.str.startswith(): Check if a string starts with a given substring.
  Example: df['A'].str.startswith('start')
- df.str.endswith(): Check if a string ends with a given substring.
  Example: df['A'].str.endswith('end')
- df.str.split(): Split each string in a column by a delimiter.
  Example: df['A'].str.split(',')

- df.str.replace(): Replace occurrences of a pattern in a string column.
  Example: df['A'].str.replace('old', 'new')
- df.str.lower(): Convert strings to lowercase.
  Example: df['A'] = df['A'].str.lower()
- df.str.upper(): Convert strings to uppercase.
  Example: df['A'] = df['A'].str.upper()
- df.str.strip(): Remove leading and trailing whitespaces.
  Example: df['A'] = df['A'].str.strip()
- df.str.len(): Get the length of each string.
  Example: df['len_A'] = df['A'].str.len()

---

11. Mathematical Operations

- df.add(), df.sub(), df.mul(), df.div(): Element-wise arithmetic operations.
  Example: df['A'] = df['A'].add(df['B'])
- df.pow(): Element-wise power function.
  Example: df['A'] = df['A'].pow(2)
- df.mod(): Element-wise modulo operation.
  Example: df['A'] = df['A'].mod(3)
- df.abs(): Absolute values of elements.
  Example: df['A'] = df['A'].abs()
- df.round(): Round numeric values to a specified number of decimal places.
  Example: df['A'] = df['A'].round(2)
- df.cumsum(): Cumulative sum of elements.
  Example: df['A_cumsum'] = df['A'].cumsum()

- df.cumprod(): Cumulative product of elements.
  Example: df['A_cumprod'] = df['A'].cumprod()
- df.cummin(): Cumulative minimum of elements.
  Example: df['A_cummin'] = df['A'].cummin()
- df.cummax(): Cumulative maximum of elements.
  Example: df['A_cummax'] = df['A'].cummax()

---

12. Statistical Methods

- df.corr(): Compute pairwise correlation of columns.
  Example: df.corr()
- df.cov(): Compute pairwise covariance of columns.
  Example: df.cov()
- df.skew(): Calculate skewness (asymmetry of the data).
  Example: df.skew()
- df.kurt(): Calculate kurtosis (tailedness of the distribution).
  Example: df.kurt()
- df.mean(), df.median(), df.mode(): Calculate basic statistics (mean, median, mode).
  Example: df['A'].mean(), df['A'].median(), df['A'].mode()
- df.min(), df.max(), df.std(), df.var(): Compute basic statistics (min, max, standard deviation, variance).
  Example: df['A'].min(), df['A'].max(), df['A'].std(), df['A'].var()
- df.cumsum(), df.cumprod(): Compute cumulative sum and product.

Example: df['A_cumsum'] = df['A'].cumsum(), df['A_cumprod'] = df['A'].cumprod()

---

13. Categorical Data

- df.astype('category'): Convert a column to categorical type.
  Example: df['A'] = df['A'].astype('category')
- df.cat.codes: Get the category codes of a categorical column.
  Example: df['A'].cat.codes
- df.cat.categories: Get the categories of a categorical column.
  Example: df['A'].cat.categories
- df.cat.rename_categories(): Rename categories in a categorical column.
  Example: df['A'] = df['A'].cat.rename_categories(['cat1', 'cat2'])

---

14. Set Operations

- df.union(): Return the union of two DataFrames.
  Example: df_union = df1.union(df2)
- df.intersection(): Return the intersection of two DataFrames.
  Example: df_intersection = df1.intersection(df2)

- df.difference(): Return the difference between two DataFrames.
  Example: df_diff = df1.difference(df2)
- df.isin(): Check if values are in a collection (list, set, etc.).
  Example: df['A'].isin([1, 2])

---

15. Data Export

- df.to_csv(): Write a DataFrame to a CSV file.
  Example: df.to_csv('output.csv', index=False)
- df.to_excel(): Write a DataFrame to an Excel file.
  Example: df.to_excel('output.xlsx', index=False)
- df.to_sql(): Write a DataFrame to a SQL database.
  Example: df.to_sql('table_name', con, if_exists='replace')
- df.to_json(): Write a DataFrame to a JSON file.
  Example: df.to_json('output.json')
- df.to_parquet(): Write a DataFrame to a Parquet file.
  Example: df.to_parquet('output.parquet')
- df.to_pickle(): Write a DataFrame to a pickle file.
  Example: df.to_pickle('output.pkl')
- df.to_html(): Write a DataFrame to an HTML table.
  Example: df.to_html('output.html')
- df.to_clipboard(): Copy the DataFrame to the system clipboard.
  Example: df.to_clipboard()

---

16. Performance Optimization

- df.query(): Query the DataFrame using a string expression.
  Example: df.query('A > 2 and B < 5')
- df.eval(): Evaluate a string expression in the context of the DataFrame.
  Example: df.eval('C = A + B')
- df.memory_usage(deep=True): Get detailed memory usage of the DataFrame.
  Example: df.memory_usage(deep=True)

---

17. Window Functions

- df.rolling(window=5): Apply rolling window functions (e.g., moving averages).
  Example: df['rolling_avg'] = df['A'].rolling(window=5).mean()
- df.expanding(): Apply expanding window functions.
  Example: df['expanding_sum'] = df['A'].expanding().sum()
- df.ewm(span=10): Apply exponentially weighted functions to time-series.
  Example: df['ewm'] = df['A'].ewm(span=10).mean()

```
pip install pandas
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\
site-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\
programdata\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\programdata\
anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in c:\users\saip5\
appdata\roaming\python\python311\site-packages (from pandas) (1.26.4)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\
lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

```
pip list
```

```
Package                        Version
------------------------------ ---------------
absl-py                        2.1.0
aiobotocore                    2.5.0
aiofiles                       22.1.0
aiohttp                        3.8.5
aioitertools                   0.7.1
aiosignal                      1.2.0
aiosqlite                      0.18.0
alabaster                      0.7.12
anaconda-anon-usage            0.4.2
anaconda-catalogs              0.2.0
anaconda-client                1.12.1
anaconda-cloud-auth            0.1.3
anaconda-navigator             2.5.0
anaconda-project               0.11.1
anyio                          3.5.0
appdirs                        1.4.4
argon2-cffi                    21.3.0
argon2-cffi-bindings           21.2.0
arrow                          1.2.3
asgiref                        3.8.1
astroid                        2.14.2
astropy                        5.1
asttokens                      2.0.5
astunparse                     1.6.3
async-timeout                  4.0.2
atomicwrites                   1.4.0
attrs                          22.1.0
Automat                        20.2.0
```

```
autopep8                          1.6.0
Babel                             2.11.0
backcall                          0.2.0
backports.functools-lru-cache     1.6.4
backports.tempfile                1.0
backports.weakref                 1.0.post1
bcrypt                            3.2.0
beautifulsoup4                    4.12.2
binaryornot                       0.4.4
black                             0.0
bleach                            4.1.0
bokeh                             3.2.1
boltons                           23.0.0
botocore                          1.29.76
Bottleneck                        1.3.5
brotlipy                          0.7.0
catboost                          1.2.7
certifi                           2023.7.22
cffi                              1.15.1
chardet                           4.0.0
charset-normalizer                2.0.4
click                             8.0.4
cloudpickle                       2.2.1
clyent                            1.2.2
colorama                          0.4.6
colorcet                          3.0.1
comm                              0.1.2
conda                             23.7.4
conda-build                       3.26.1
conda-content-trust               0.2.0
conda_index                       0.3.0
conda-libmamba-solver             23.7.0
conda-pack                        0.6.0
conda-package-handling            2.2.0
conda_package_streaming           0.9.0
conda-repo-cli                    1.0.75
conda-token                       0.4.0
conda-verify                      3.4.2
constantly                        15.1.0
contourpy                         1.0.5
cookiecutter                      1.7.3
cryptography                      41.0.3
cssselect                         1.1.0
cycler                            0.11.0
cytoolz                           0.12.0
daal4py                           2023.1.1
dask                              2023.6.0
datasets                          2.12.0
datashader                        0.15.2
```

```
datashape                   0.5.4
debugpy                     1.6.7
decorator                   5.1.1
defusedxml                  0.7.1
diff-match-patch            20200713
dill                        0.3.6
distributed                 2023.6.0
Django                      5.1.5
docstring-to-markdown       0.11
docutils                    0.18.1
entrypoints                 0.4
et-xmlfile                  1.1.0
executing                   0.8.3
fastjsonschema              2.16.2
filelock                    3.9.0
flake8                      6.0.0
Flask                       2.2.2
flatbuffers                 24.3.25
fonttools                   4.25.0
frozenlist                  1.3.3
fsspec                      2023.4.0
future                      0.18.3
gast                        0.6.0
gensim                      4.3.0
glob2                       0.7
google-pasta                0.2.0
graphviz                    0.20.3
greenlet                    2.0.1
grpcio                      1.68.1
h5py                        3.12.1
HeapDict                    1.0.1
holoviews                   1.17.1
huggingface-hub             0.15.1
hvplot                      0.8.4
hyperlink                   21.0.0
idna                        3.4
imagecodecs                 2023.1.23
imageio                     2.26.0
imagesize                   1.4.1
imbalanced-learn            0.10.1
importlib-metadata          6.0.0
incremental                 21.3.0
inflection                  0.5.1
iniconfig                   1.1.1
intake                      0.6.8
intervaltree                3.1.0
ipykernel                   6.25.0
ipython                     8.15.0
ipython-genutils            0.2.0
```

```
ipywidgets                    8.0.4
isort                         5.9.3
itemadapter                   0.3.0
itemloaders                   1.0.4
itsdangerous                  2.0.1
jaraco.classes                3.2.1
jedi                          0.18.1
jellyfish                     1.0.1
Jinja2                        3.1.2
jinja2-time                   0.2.0
jmespath                      0.10.0
joblib                        1.2.0
json5                         0.9.6
jsonpatch                     1.32
jsonpointer                   2.1
jsonschema                    4.17.3
jupyter                       1.0.0
jupyter_client                7.4.9
jupyter-console               6.6.3
jupyter_core                  5.3.0
jupyter-events                0.6.3
jupyter-server                1.23.4
jupyter_server_fileid         0.9.0
jupyter_server_ydoc           0.8.0
jupyter-ydoc                  0.2.4
jupyterlab                    3.6.3
jupyterlab-pygments           0.1.2
jupyterlab_server             2.22.0
jupyterlab-visualpython       3.0.2
jupyterlab-widgets            3.0.5
kaleido                       0.2.1
keras                         3.7.0
keyring                       23.13.1
kiwisolver                    1.4.4
lazy_loader                   0.2
lazy-object-proxy             1.6.0
libarchive-c                  2.9
libclang                      18.1.1
libmambapy                    1.5.1
lightgbm                      4.5.0
linkify-it-py                 2.0.0
llvmlite                      0.40.0
lmdb                          1.4.1
locket                        1.0.0
lxml                          4.9.3
lz4                           4.3.2
Markdown                      3.4.1
markdown-it-py                2.2.0
MarkupSafe                    2.1.1
```

```
matplotlib                    3.7.2
matplotlib-inline             0.1.6
mccabe                        0.7.0
mdit-py-plugins               0.3.0
mdurl                         0.1.0
menuinst                      1.4.19
mistune                       0.8.4
mkl-fft                       1.3.8
mkl-random                    1.2.4
mkl-service                   2.4.0
ml-dtypes                     0.4.1
more-itertools                8.12.0
mpmath                        1.3.0
msgpack                       1.0.3
multidict                     6.0.2
multipledispatch              0.6.0
multiprocess                  0.70.14
munkres                       1.1.4
mypy-extensions               1.0.0
namex                         0.0.8
navigator-updater             0.4.0
nbclassic                     0.5.5
nbclient                      0.5.13
nbconvert                     6.5.4
nbformat                      5.9.2
nest-asyncio                  1.5.6
networkx                      3.1
nltk                          3.8.1
notebook                      6.5.4
notebook_shim                 0.2.2
numba                         0.57.1
numexpr                       2.8.4
numpy                         1.26.4
numpydoc                      1.5.0
opencv-python                 4.11.0.86
openpyxl                      3.0.10
opt_einsum                    3.4.0
optree                        0.13.1
packaging                     23.1
pandas                        2.0.3
pandocfilters                 1.5.0
panel                         1.2.3
param                         1.13.0
paramiko                      2.8.1
parsel                        1.6.0
parso                         0.8.3
partd                         1.4.0
pathlib                       1.0.1
pathspec                      0.10.3
```

```
patsy                         0.5.3
pep8                          1.7.1
pexpect                       4.8.0
pickleshare                   0.7.5
Pillow                        9.4.0
pip                           23.2.1
pkce                          1.0.3
pkginfo                       1.9.6
platformdirs                  3.10.0
plotly                        5.9.0
pluggy                        1.0.0
ply                           3.11
poyo                          0.5.0
prometheus-client             0.14.1
prompt-toolkit                3.0.36
Protego                       0.1.16
protobuf                      5.29.0
psutil                        5.9.0
ptyprocess                    0.7.0
pure-eval                     0.2.2
py-cpuinfo                    8.0.0
pyarrow                       11.0.0
pyasn1                        0.4.8
pyasn1-modules                0.2.8
pycodestyle                   2.10.0
pycosat                       0.6.4
pycparser                     2.21
pyct                          0.5.0
pycurl                        7.45.2
pydantic                      1.10.8
PyDispatcher                  2.0.5
pydocstyle                    6.3.0
pyerfa                        2.0.0
pyflakes                      3.0.1
Pygments                      2.15.1
PyJWT                         2.4.0
pylint                        2.16.2
pylint-venv                   2.3.0
pyls-spyder                   0.4.0
PyNaCl                        1.5.0
pyodbc                        4.0.34
pyOpenSSL                     23.2.0
pyparsing                     3.0.9
pyperclip                     1.9.0
PyQt5                         5.15.7
PyQt5-sip                     12.11.0
PyQtWebEngine                 5.15.4
pyrsistent                    0.18.0
PySocks                       1.7.1
```

pytest	7.4.0
python-dateutil	2.8.2
python-dotenv	0.21.0
python-json-logger	2.0.7
python-lsp-black	1.2.1
python-lsp-jsonrpc	1.0.0
python-lsp-server	1.7.2
python-slugify	5.0.2
python-snappy	0.6.1
pytoolconfig	1.2.5
pytz	2023.3.post1
pyviz-comms	2.3.0
PyWavelets	1.4.1
pywin32	305.1
pywin32-ctypes	0.2.0
pywinpty	2.0.10
PyYAML	6.0
pyzmq	23.2.0
QDarkStyle	3.0.2
qstylizer	0.2.2
QtAwesome	1.2.2
qtconsole	5.4.2
QtPy	2.2.0
queuelib	1.5.0
readchar	4.2.1
regex	2022.7.9
requests	2.31.0
requests-file	1.5.1
requests-toolbelt	1.0.0
responses	0.13.3
rfc3339-validator	0.1.4
rfc3986-validator	0.1.1
rich	13.9.4
rope	1.7.0
Rtree	1.0.1
ruamel.yaml	0.17.21
ruamel-yaml-conda	0.17.21
s3fs	2023.4.0
safetensors	0.3.2
scikit-image	0.20.0
scikit-learn	1.3.0
scikit-learn-intelex	20230426.121932
scipy	1.11.1
Scrapy	2.8.0
seaborn	0.12.2
Send2Trash	1.8.0
service-identity	18.1.0
setuptools	68.0.0
sip	6.6.2

```
six                               1.16.0
smart-open                        5.2.1
sniffio                           1.2.0
snowballstemmer                   2.2.0
sortedcontainers                  2.4.0
soupsieve                         2.4
Sphinx                            5.0.2
sphinxcontrib-applehelp           1.0.2
sphinxcontrib-devhelp             1.0.2
sphinxcontrib-htmlhelp            2.0.0
sphinxcontrib-jsmath              1.0.1
sphinxcontrib-qthelp              1.0.3
sphinxcontrib-serializinghtml     1.1.5
spyder                            5.4.3
spyder-kernels                    2.4.4
SQLAlchemy                        1.4.39
sqlparse                          0.5.3
stack-data                        0.2.0
statsmodels                       0.14.0
sympy                             1.13.1
tables                            3.8.0
tabulate                          0.8.10
TBB                               0.2
tblib                             1.7.0
tenacity                          8.2.2
tensorboard                       2.18.0
tensorboard-data-server           0.7.2
tensorflow                        2.18.0
tensorflow_intel                  2.18.0
tensorflow-io-gcs-filesystem      0.31.0
termcolor                         2.5.0
terminado                         0.17.1
text-unidecode                    1.3
textdistance                      4.2.1
threadpoolctl                     2.2.0
three-merge                       0.1.1
tifffile                          2023.4.12
tinycss2                          1.2.1
tldextract                        3.2.0
tokenizers                        0.13.2
toml                              0.10.2
tomlkit                           0.11.1
toolz                             0.12.0
torch                             2.5.1
torchvision                       0.20.1
tornado                           6.3.2
tqdm                              4.65.0
traitlets                         5.7.1
transformers                      4.32.1
```

```
Twisted                       22.10.0
twisted-iocpsupport           1.0.2
typing_extensions             4.12.2
tzdata                        2023.3
uc-micro-py                   1.0.1
ujson                         5.4.0
Unidecode                     1.2.0
urllib3                       1.26.16
VisualPy                      1.0.1
visualpython                  3.0.2
w3lib                         1.21.0
watchdog                      2.1.6
wcwidth                       0.2.5
webencodings                  0.5.1
websocket-client              0.58.0
Werkzeug                      2.2.3
whatthepatch                  1.0.2
wheel                         0.38.4
widgetsnbextension            4.0.5
win-inet-pton                 1.1.0
wrapt                         1.14.1
xarray                        2023.6.0
xgboost                       2.1.3
xlwings                       0.29.1
xxhash                        2.0.2
xyzservices                   2022.9.0
y-py                          0.5.9
yapf                          0.31.0
yarl                          1.8.1
ypy-websocket                 0.8.2
zict                          2.2.0
zipp                          3.11.0
zope.interface                5.4.0
zstandard                     0.19.0
Note: you may need to restart the kernel to use updated packages.
```

```python
import pandas as pd
print(pd.__version__)
```

```
2.0.3
```

```python
df=pd.DataFrame({'A':[1,2,3,4],'B':[2,4,6,7]})
print(df)
```

```
   A  B
0  1  2
1  2  4
2  3  6
3  4  7
```

```
data=[[1,'temp1'],[23,'temp2']]
df=pd.DataFrame(columns=['ID','Name'],
index=['row1','row2'],data=data)
print(df)
```

```
      ID   Name
row1   1  temp1
row2  23  temp2
```

```
s=pd.Series([1,2,2,2,None,2,2])
print(s)
```

```
0    1.0
1    2.0
2    2.0
3    2.0
4    NaN
5    2.0
6    2.0
dtype: float64
```

```
s=pd.Series([1,2,2,2,None,2,2],index=['temp'+str(i) for i in
range(7)])
print(s)
```

```
temp0    1.0
temp1    2.0
temp2    2.0
temp3    2.0
temp4    NaN
temp5    2.0
temp6    2.0
dtype: float64
```

join: How to handle indexes on the other axis: 'outer' (default): Union of the columns/rows (like a full outer join). 'inner': Intersection of the columns/rows (like an inner join).

```
df1=pd.DataFrame({'A':[1,2]})
df2=pd.DataFrame({'A':[3,5]})
df_combined=pd.concat([df1,df2], axis=1)  #syntax: pd.concat(obj,
axis=0/1, join='outer')
print(df_combined)
```

```
   A  A
0  1  3
1  2  5
```

```
import pandas as pd

df1 = pd.DataFrame({
    'A': [1, 2],
```

```python
    'B': [3, 4]
})
df2 = pd.DataFrame({
    'B': [5, 6],
    'C': [7, 8]
})
# Concatenate with join='outer' (default)
result = pd.concat([df1, df2], axis=0, join='outer')
print(result)
```

```
     A  B    C
0  1.0  3  NaN
1  2.0  4  NaN
0  NaN  5  7.0
1  NaN  6  8.0
```

```python
result_inner = pd.concat([df1, df2], axis=0, join='inner')
print(result_inner)
```

```
   B
0  3
1  4
0  5
1  6
```

```python
df1 = pd.DataFrame({'A': [1], 'B': [5]})
df2 = pd.DataFrame({'A': [2], 'C': [6]})
df_combined=pd.concat([df1,df2],axis=1,join='outer')
print(df_combined)
```

```
   A  B  A  C
0  1  5  2  6
```

```python
#pd.merge() combines dataframe using sql style joins (eg., inner,
outer, left, right) based on the key
df1 = pd.DataFrame({'key': ['K0', 'K1'], 'A': [1, 2]})
df2 = pd.DataFrame({'key': ['K0', 'K1'], 'B': [3, 4]})
df_merged=pd.merge(df1,df2, on='key',how='right') #pd.merge(left,
right, on=None, how='inner')
print(df_merged)
```

```
  key  A  B
0  K0  1  3
1  K1  2  4
```

```python
df1 = pd.DataFrame({'id': [1, 2], 'name': ['Alice', 'Bob']})
df2 = pd.DataFrame({'id': [2, 3], 'score': [85, 90]})
df_merged=pd.merge(df1,df2, on='id', how='outer')
print(df_merged)
```

```
   id   name  score
0   1  Alice    NaN
1   2    Bob   85.0
2   3    NaN   90.0

df=pd.read_csv('diabetes.csv') #pd.read_csv(filename, delimiter=',')
print(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
None

import pandas as pd
df = pd.read_excel('data.xlsx', sheet_name='Sheet1')
print(df)

import pandas as pd
import sqlite3
con=sqlite3.connect('mydatabase.db')
df=df.read_sql('select * from my_table', con)
print(df)

df=pd.read_json('data.json')
print(df)

df=pd.read_parquet('userdata1.parquet')
print(df)

df=pd.read_csv('data.html')
print(df)

df=pd.read_pickle('data.pkl')
print(df)

#to remove overlapping columns
df1 = pd.DataFrame({'key': ['A', 'B'], 'value': [1, 2]})
df2 = pd.DataFrame({'key': ['A', 'B'], 'value': [3, 4]})
```

```
df_merged = pd.merge(df1, df2, on='key', suffixes=('_left', '_right'))
print(df_merged)
```

```
  key  value_left  value_right
0   A           1            3
1   B           2            4
```

this section covers methods to explore and understand your DataFrame--its structure, contents, and basic properties. inspecting it is crucial to understand what you're working with—its size, data types, missing values, and a preview of the data.

```
#df.head(n) it is going to display first n rows of data
df=pd.read_parquet('userdata1.parquet')
print(df.head(2))

print(df.tail(2))
```

```
  Category Type  Value  Score
3        B    Y     40     80
4        A    X     15     88
```

```
#df.shape is going to return a tuple with the number of rows and columns.
print(df.shape) #indicating that 1000rows , 13columns
```

```
(5, 4)
```

```
df.info() #df.info() provides a summary of the data frame(including column name, data types, non-null counts)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Category  5 non-null      object
 1   Type      5 non-null      object
 2   Value     5 non-null      int64
 3   Score     5 non-null      int64
dtypes: int64(2), object(2)
memory usage: 292.0+ bytes
```

```
df.describe() #df.describe generates summary statistics(count, mean, etc) for numarical columns
```

```
           Value     Score
count   5.000000   5.00000
mean   23.000000  87.60000
std    12.041595   5.59464
min    10.000000  80.00000
25%    15.000000  85.00000
```

```
50%    20.000000  88.00000
75%    30.000000  90.00000
max    40.000000  95.00000
```

```python
df.columns #df.columns returns the column names as an index object
```

```
Index(['Category', 'Type', 'Value', 'Score'], dtype='object')
```

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

```
['Category', 'Type', 'Value', 'Score']
```

```python
df.dtypes #df.dtypes returns the data type of each coloumn
```

```
Category    object
Type        object
Value        int64
Score        int64
dtype: object
```

```python
df.isnull() #df.isnull() returns a boolean data frame showing True for missing valeus (NaN)!
```

```
   Category   Type  Value  Score
0     False  False  False  False
1     False  False  False  False
2     False  False  False  False
3     False  False  False  False
4     False  False  False  False
```

```python
df.isnull().sum() #to count each column missing values use df.isnull().sum()
```

```
Category    0
Type        0
Value       0
Score       0
dtype: int64
```

```python
df.notnull()
```

```
   Category  Type  Value  Score
0      True  True   True   True
1      True  True   True   True
2      True  True   True   True
3      True  True   True   True
4      True  True   True   True
```

```python
df.notnull().sum()
```

```
Category    5
Type        5
```

```
Value       5
Score       5
dtype: int64

df.notnull().all(axis=1).sum()

5

df.notnull().all(axis=0).sum()

4

#df.memory_usage() returns memory usage(in bytes) for each column.
df.memory_usage(deep=True) #deep=Trye gives more accurate count for
object types (like strings)

df.isnull().mean()*100

#practice
import pandas as pd

# Create a varied DataFrame
df = pd.DataFrame({
    'ID': [1, 2, None, 4],
    'Name': ['Alice', 'Bob', None, 'David'],
    'Score': [85.5, None, 90.0, 95.0],
    'Date': pd.to_datetime(['2023-01-01', None, '2023-01-03', '2023-
01-04'])
})

# Explore it
print("Head (3):\n", df.head(3))
print("\nTail (2):\n", df.tail(2))
print("\nShape:", df.shape)
print("\nInfo:")
df.info()
print("\nDescribe:\n", df.describe())
print("\nColumns:", list(df.columns))
print("\nDtypes:\n", df.dtypes)
print("\nNull counts:\n", df.isnull().sum())
print("\nMemory:\n", df.memory_usage(deep=True))

#df.head(n) it is going to display first n rows of data
df=pd.read_parquet('userdata1.parquet')
print(df.head(2))
```

This section focuses on how to access, extract, and manipulate specific parts of a DataFrame using various indexing techniques. access columns, rows, or individual values using labels, positions, or conditions.

```python
#df['column_name'] to access a specific column as a Series
df['title']

#instead of that we can even call df.column_name as well
df.title

#df.iloc[] indexes by integer position (row, column).
df.iloc[0:2,1:10] #syntax: df.iloc[row_index, column_index]

#df.loc[] indexes by label (row/column names)
df.loc[100,'salary'] #df.loc[row_label, column_label]

#instead fo df.loc[] we can use df.at[row_label, column_label] for
single value access it is even more faster!
df.at[10,'gender']

#df.iat[] Accesses a single value by integer position (faster than
iloc for scalars).
df.iat[10,3]



#df.query() filters rows using a string expression!
df.query("gender=='Female' and salary>10000")

print(df.xs(0))

df.iloc[0,0:]

df.set_index('id', inplace=True) #sets one or more columns as the
index.
print(df.index) # inplace=True modifies the DataFrame directly.

df.reset_index(inplace=True) #undo the index
print(df.index)

df1 = pd.DataFrame({'B': [4, 5, 6]}, index=[2, 0, 1])
print(df1.sort_index()) #syntax: df.sort_index(axis=0, ascending=True)

print(df1.sort_index(ascending=False))

#practice
import pandas as pd

# Create a DataFrame
df1 = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'Score': [85, 90, 95]
}, index=['x', 'y', 'z'])

# Selection and Indexing
print("Column 'Name':\n", df1['Name'])
```

```
print("\nRow 1 (iloc):\n", df1.iloc[1])
print("\nRow 'y' (loc):\n", df1.loc['y'])
print("\nValue at 'x', 'Age':", df1.at['x', 'Age'])
print("\nValue at position (0, 2):", df1.iat[0, 2])
print("\nQuery Age > 25:\n", df1.query('Age > 25'))
print("\nCross-section 'z':\n", df1.xs('z'))
df1.set_index('Name', inplace=True)
print("\nAfter set_index:\n", df1)
df1.reset_index(inplace=True)
print("\nAfter reset_index:\n", df1)
print("\nSorted by index:\n", df1.sort_index())
```

```
Column 'Name':
 x      Alice
y        Bob
z    Charlie
Name: Name, dtype: object

Row 1 (iloc):
 Name     Bob
Age       30
Score     90
Name: y, dtype: object

Row 'y' (loc):
 Name     Bob
Age       30
Score     90
Name: y, dtype: object

Value at 'x', 'Age': 25

Value at position (0, 2): 85

Query Age > 25:
       Name  Age  Score
y      Bob   30     90
z  Charlie   35     95

Cross-section 'z':
 Name     Charlie
Age           35
Score         95
Name: z, dtype: object

After set_index:
          Age  Score
Name
Alice     25     85
Bob       30     90
```

```
Charlie   35     95

After reset_index:
      Name  Age  Score
0    Alice   25     85
1      Bob   30     90
2  Charlie   35     95

Sorted by index:
      Name  Age  Score
0    Alice   25     85
1      Bob   30     90
2  Charlie   35     95
```

This section covers methods to handle missing values, duplicates, and transformations to prepare your data frame for anayalysis.

```
#df.dropna() removes rows or columns with missing values (NaN)
temp=pd.DataFrame({'A': [1, None, 3], 'B': [4, 5, 6]})
print(temp.info())
print(temp.dropna(how='any')) #syntax: df.dropna(axis=0,
how='any/all', inplace=False)
print()
print(temp)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       2 non-null      float64
 1   B       3 non-null      int64
dtypes: float64(1), int64(1)
memory usage: 180.0 bytes
None
     A  B
0  1.0  4
2  3.0  6

     A  B
0  1.0  4
1  NaN  5
2  3.0  6

df1 = pd.DataFrame({'A': [None, None], 'B': [None, 1]})
print(df1.dropna(how='all')) #drop if all values are missing

      A    B
1  None  1.0
```

```python
#df.fillna()) fills missing values with a specific value or method!
print(df1.fillna(0)) #df.fillna(value, method=None, inplace=False)
```

```
   A    B
0  0  0.0
1  0  1.0
```

```python
df1['B'].fillna(df1['B'].mean())
```

```
0    1.0
1    1.0
Name: B, dtype: float64
```

```python
k=df.fillna(method='ffill') #methods are ffill or bfill means forward fill and backward fill
k.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
```

```python
#df.replace() replaces specific value with others.
import numpy as np
df1.replace(0,np.nan) #syntax: df.replace(to_place, value, inplace=False)
```

```
      A    B
0  None  NaN
1  None  1.0
```

```python
df1 = pd.DataFrame({'A': range(100), 'B': range(2000,2100)})
df1.replace([0,1],[100000,2000000],inplace=True)    #replace multiple values!
df1
```

```
          A     B
0    100000  2000
1   2000000  2001
```

```
2          2  2002
3          3  2003
4          4  2004
..       ...   ...
95        95  2095
96        96  2096
97        97  2097
98        98  2098
99        99  2099

[100 rows x 2 columns]

print(df1.replace({'A': 3, 'B': 1}, 999)) #using dictionary

           A     B
0     100000  2000
1    2000000  2001
2          2  2002
3        999  2003
4          4  2004
..       ...   ...
95        95  2095
96        96  2096
97        97  2097
98        98  2098
99        99  2099

[100 rows x 2 columns]

print(df1.drop('A',axis=1)) #df.drop(labels, axis=0, inplace=False)
used to drop specific row or columns.

       B
0   2000
1   2001
2   2002
3   2003
4   2004
..   ...
95  2095
96  2096
97  2097
98  2098
99  2099

[100 rows x 1 columns]

df1.drop([0,1,6]) #drop specific row indexes

    A     B
2   2  2002
```

```
3    3  2003
4    4  2004
5    5  2005
7    7  2007
..  ..   ...
95  95  2095
96  96  2096
97  97  2097
98  98  2098
99  99  2099

[97 rows x 2 columns]
```

```python
df1.drop(df1[df1['A']==100000].index, axis=0)
```

```
          A     B
1   2000000  2001
2         2  2002
3         3  2003
4         4  2004
5         5  2005
..      ...   ...
95       95  2095
96       96  2096
97       97  2097
98       98  2098
99       99  2099

[99 rows x 2 columns]
```

```python
df1 = df1.loc[df1['A'] != 100000] #filter and drop
df1
```

```
          A     B
1   2000000  2001
2         2  2002
3         3  2003
4         4  2004
5         5  2005
..      ...   ...
95       95  2095
96       96  2096
97       97  2097
98       98  2098
99       99  2099

[99 rows x 2 columns]
```

```python
df1.rename(columns={'A':'new_a'},inplace=True)
df1.columns
```

```
C:\Users\saip5\AppData\Local\Temp\ipykernel_12416\157376447.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df1.rename(columns={'A':'new_a'},inplace=True)

Index(['new_a', 'B'], dtype='object')
```

```
print(df1.rename(columns=str.upper))
```

```
      NEW_A     B
1   2000000  2001
2         2  2002
3         3  2003
4         4  2004
5         5  2005
..      ...   ...
95       95  2095
96       96  2096
97       97  2097
98       98  2098
99       99  2099

[99 rows x 2 columns]
```

```
df1.dtypes
```

```
new_a    int64
B        int64
dtype: object
```

```
df1['new_a']=df1['new_a'].astype(float)
df1.dtypes
```

```
C:\Users\saip5\AppData\Local\Temp\ipykernel_12416\1067428253.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df1['new_a']=df1['new_a'].astype(float)

new_a    float64
B          int64
dtype: object
```

```
df1 = pd.DataFrame({'A': [1, 1, 2], 'B': [3, 3, 4]})
df1.duplicated() #identifies duplicate rows

0    False
1     True
2    False
dtype: bool

df1

   A  B
0  1  3
1  1  3
2  2  4

df1.drop_duplicates(keep='last')

   A  B
1  1  3
2  2  4

df1['A'].isin([1,2]).sum()

3

# df1.str.* applies string methods to text column
df = pd.DataFrame({'A': ['ABC', 'DEF', 'GHI']})
df['A']=df['A'].str.lower()
df['A'].str[0:2]

0    ab
1    de
2    gh
Name: A, dtype: object

# practice
import pandas as pd
import numpy as np

# Create a messy DataFrame
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', None, 'Bob'],
    'Age': [25, None, 30, 25],
    'Score': [85, 90, None, 90],
    'Text': ['UPPER', 'lower', 'Mixed', 'UPPER']
})

# Clean it
print("Original:\n", df)
print("\nDrop NA rows:\n", df.dropna())
print("\nFill NA with 0:\n", df.fillna(0))
print("\nReplace None with 'Unknown':\n", df.replace({None:
```

```
'Unknown'}))
df.drop('Score', axis=1, inplace=True)
print("\nDrop 'Score' column:\n", df)
df.rename(columns={'Age': 'Years'}, inplace=True)
print("\nRename 'Age' to 'Years':\n", df)
df['Years'] = df['Years'].astype('Float64')
print("\nYears as Float64:\n", df)
print("\nDuplicates:\n", df.duplicated())
df.drop_duplicates(inplace=True)
print("\nDrop duplicates:\n", df)
print("\nNames in ['Alice', 'Bob']:\n", df['Name'].isin(['Alice',
'Bob']))
df['Text'] = df['Text'].str.lower()
print("\nText to lowercase:\n", df)
```

```
Original:
    Name   Age  Score   Text
0  Alice  25.0   85.0  UPPER
1    Bob   NaN   90.0  lower
2   None  30.0    NaN  Mixed
3    Bob  25.0   90.0  UPPER

Drop NA rows:
    Name   Age  Score   Text
0  Alice  25.0   85.0  UPPER
3    Bob  25.0   90.0  UPPER

Fill NA with 0:
    Name   Age  Score   Text
0  Alice  25.0   85.0  UPPER
1    Bob   0.0   90.0  lower
2      0  30.0    0.0  Mixed
3    Bob  25.0   90.0  UPPER

Replace None with 'Unknown':
      Name   Age  Score   Text
0    Alice  25.0   85.0  UPPER
1      Bob   NaN   90.0  lower
2  Unknown  30.0    NaN  Mixed
3      Bob  25.0   90.0  UPPER

Drop 'Score' column:
    Name   Age   Text
0  Alice  25.0  UPPER
1    Bob   NaN  lower
2   None  30.0  Mixed
3    Bob  25.0  UPPER

Rename 'Age' to 'Years':
    Name  Years   Text
```

```
0   Alice   25.0  UPPER
1     Bob    NaN  lower
2    None   30.0  Mixed
3     Bob   25.0  UPPER

Years as Float64:
     Name  Years   Text
0   Alice   25.0  UPPER
1     Bob   <NA>  lower
2    None   30.0  Mixed
3     Bob   25.0  UPPER

Duplicates:
 0    False
1    False
2    False
3    False
dtype: bool

Drop duplicates:
     Name  Years   Text
0   Alice   25.0  UPPER
1     Bob   <NA>  lower
2    None   30.0  Mixed
3     Bob   25.0  UPPER

Names in ['Alice', 'Bob']:
 0     True
1     True
2    False
3     True
Name: Name, dtype: bool

Text to lowercase:
     Name  Years   Text
0   Alice   25.0  upper
1     Bob   <NA>  lower
2    None   30.0  mixed
3     Bob   25.0  upper
```

These are Powerful tools for summarizing data, calculating statistics, and exploring relationships within your dataset.

```python
import pandas as pd
df = pd.DataFrame({'A': ['foo', 'bar', 'foo', 'bar'], 'B': [1, 2, 3,
4]})
print(df.groupby('A').sum())
```

```
     B
A
```

```
bar  6
foo  4

df = pd.DataFrame({
    'Category': ['A', 'A', 'B', 'B'],
    'Type': ['X', 'Y', 'X', 'Y'],
    'Value': [10, 20, 30, 40]
})
print(df.groupby(['Category', 'Type']).sum()) #multiple columns

               Value
Category Type
A        X        10
         Y        20
B        X        30
         Y        40

df.groupby('Category').count()

          Type  Value
Category
A            2      2
B            2      2

#df.agg() applies one or more aggregation functions to grouped or
ungrouped data.
df.groupby('Category').agg({'Type':'count','Value':'mean'})

          Type  Value
Category
A            2   15.0
B            2   35.0

df = pd.DataFrame({'A': ['foo', 'bar', 'foo', 'bar'], 'B': [1, 12, 3,
64]})
def range_function(x):
    return x.max()-x.min()
print(df.groupby('A').agg({'B':range_function}))

      B
A
bar  52
foo   2

#syntax: df.pivote_table(values,index, columns=None, aggfunc='mean')

print(df.pivot_table(values='B', index='A',
aggfunc=['sum','mean','std','var','max','min']))

    sum mean        std   var max min
      B    B          B     B   B   B
A
```

```
bar  76   38  36.769553  1352  64  12
foo   4    2   1.414214     2   3   1
```

```python
#df.crosstab() computes a cross-tabulation of two or more
factors( like a frequency table)
df = pd.DataFrame({'A': ['foo', 'foo', 'bar'], 'B': ['x', 'y', 'x']})
pd.crosstab(df['A'],df['B'])
''' is useful when you want to summarize categrical data and
understand relationship between two or more categorical
variables '''
```

```
' is useful when you want to summarize categrical data and understand
relationship between two or more categorical \nvariables '
```

```python
df = pd.DataFrame({
    'Department': ['HR', 'Finance', 'HR', 'IT', 'Finance'],
    'Gender': ['Male', 'Female', 'Female', 'Male', 'Male'],
    'Salary': [50000, 60000, 55000, 70000, 80000]
})

# Aggregating salaries by department and gender
print(pd.crosstab(df['Department'], df['Gender'], values=df['Salary'],
aggfunc='mean'))
```

```
Gender       Female     Male
Department
Finance     60000.0  80000.0
HR          55000.0  50000.0
IT              NaN  70000.0
```

```python
df = pd.DataFrame({
    'Gender': ['M', 'F', 'M', 'F'],
    'Pass': ['Yes', 'No', 'Yes', 'No']
})
print(pd.crosstab(df['Gender'], df['Pass']))
```

```
Pass    No  Yes
Gender
F        2    0
M        0    2
```

```python
#basic statistics
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
print(df['A'].mean())
print(df['A'].median())
print(df['A'].mode()) #aggfunc is typically used with functions like
crosstab() or pivot_table()
print(df['B'].agg(['mean', 'median', 'min', 'max']))
```

```
2.0
2.0
```

```
0    1
1    2
2    3
Name: A, dtype: int64
mean      5.0
median    5.0
min       4.0
max       6.0
Name: B, dtype: float64
```

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
print(df['A'].mean())
```

```
2.0
```

```
print(df['A'].sum())
```

```
6
```

```
print(df.max())
```

```
A    3
B    6
dtype: int64
```

```
print(df.std())
```

```
A    1.0
B    1.0
dtype: float64
```

```
df = pd.DataFrame({'A': [1, None, 3], 'B': [4, 5, 6]})
print(df['A'].count())
```

```
2
```

```
df = pd.DataFrame({
    'Group': ['X', 'X', 'Y'],
    'Value': [10, 20, 30]
})
print(df.groupby('Group').last())
```

```
       Value
Group
X         20
Y         30
```

```
df = pd.DataFrame({'A': ['foo', 'bar', 'foo'], 'B': [1, 2, 3]})
print(df.groupby('A').first())
```

```
     B
A
```

```
bar  2
foo  1

#practice
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({
    'Category': ['A', 'A', 'B', 'B', 'A'],
    'Type': ['X', 'Y', 'X', 'Y', 'X'],
    'Value': [10, 20, 30, 40, 15],
    'Score': [85, 90, 95, 80, 88]
})

# Aggregation and Grouping
print("Group by Category (sum):\n", df.groupby('Category').sum())
print("\nAgg with multiple functions:\n",
df.groupby('Category').agg({'Value': 'sum', 'Score': 'mean'}))
print("\nPivot table:\n", df.pivot_table(values='Value',
index='Category', columns='Type', aggfunc='mean'))
print("\nCrosstab:\n", pd.crosstab(df['Category'], df['Type']))
print("\nMean of Value:", df['Value'].mean())
print("\nMax per column:\n", df.max())
print("\nCount non-null:\n", df.count())
print("\nFirst in each group:\n", df.groupby('Category').first())

Group by Category (sum):
          Type  Value  Score
Category
A          XYX     45    263
B           XY     70    175

Agg with multiple functions:
           Value      Score
Category
A             45  87.666667
B             70  87.500000

Pivot table:
 Type         X     Y
Category
A          12.5  20.0
B          30.0  40.0

Crosstab:
 Type      X  Y
Category
A          2  1
B          1  1
```

```
Mean of Value: 23.0

Max per column:
 Category     B
Type         Y
Value       40
Score       95
dtype: object

Count non-null:
 Category    5
Type        5
Value       5
Score       5
dtype: int64

First in each group:
          Type  Value  Score
Category
A            X     10     85
B            X     30     95
```

Combining datasets is a common task in data analysis. These methods allow you to integrate data from different souces based on rows, columns, or specific keys.

```
import pandas as pd
df1 = pd.DataFrame({'key': ['K0', 'K2'], 'A': [1, 2]})
df2 = pd.DataFrame({'key': ['K0', 'K1'], 'B': [3, 4]})
df_merged=pd.merge(df1,df2, on='key',how='right') #df.merge(right,
on='colum_in_same',how='inner/outer/righ/left',suffixes=('_x','_y'))
print(df_merged)

  key    A  B
0  K0  1.0  3
1  K1  NaN  4

df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['Alice', 'Bob',
'Charlie']})
df2 = pd.DataFrame({'ID': [2, 4], 'Score': [90, 85]})
print(pd.merge(df1, df2, on='ID', how='left'))

   ID     Name  Score
0   1    Alice    NaN
1   2      Bob   90.0
2   3  Charlie    NaN

df1 = pd.DataFrame({'key': ['A', 'B'], 'value': [1, 2]})
df2 = pd.DataFrame({'key': ['A', 'C'], 'value': [3, 4]})
print(pd.merge(df1, df2, on='key', how='outer', suffixes=('_left',
'_right')))
```

```
  key  value_left  value_right
0   A         1.0          3.0
1   B         2.0          NaN
2   C         NaN          4.0

df1 = pd.DataFrame({'key': ['A', 'B'], 'value': [1, 2]})
df2 = pd.DataFrame({'key': ['A', 'C'], 'value': [3, 4]})

pd.concat([df1,df2],axis=0)

  key  value
0   A      1
1   B      2
0   A      3
1   C      4

df1 = pd.DataFrame({'A': [1, 2]}, index=['x', 'y'])
df2 = pd.DataFrame({'B': [3, 4]}, index=['y', 'z'])
print(df1.join(df2,how="left"))

   A    B
x  1  NaN
y  2  3.0

pd.concat([df1,df2],axis=0)

     A    B
x  1.0  NaN
y  2.0  NaN
y  NaN  3.0
z  NaN  4.0
```

It helps you organize your data for analysis, visualization, or reporting. these methods allow you to reorder rows or columns.

```
import pandas as pd
df=pd.DataFrame({'A': [3, 1, 2], 'B': [6, 4, 5]})
df.sort_values('A', ascending=False,inplace=True)
df

   A  B
0  3  6
2  2  5
1  1  4

df.sort_values(['A','B'],ascending=[True,False])

   A  B
1  1  4
2  2  5
0  3  6
```

```python
df.sort_index(ascending=False)
```

	A	B
2	2	5
1	1	4
0	3	6

```python
df.sort_index(ascending=True, inplace=True)
df
```

	A	B
0	3	6
1	1	4
2	2	5

```python
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank'],
    'Score': [85, 92, 88, 92, 79, 88]
}
df=pd.DataFrame(data)
df.sort_values('Score',inplace=True)
```

```python
df['average_rank']=df['Score'].rank(method='average')
df
```

	Name	Score	average_rank
4	Eve	79	1.0
0	Alice	85	2.0
2	Charlie	88	3.5
5	Frank	88	3.5
1	Bob	92	5.5
3	David	92	5.5

```python
df['min_rank']=df['Score'].rank(method='min')
df
```

	Name	Score	average_rank	min_rank
4	Eve	79	1.0	1.0
0	Alice	85	2.0	2.0
2	Charlie	88	3.5	3.0
5	Frank	88	3.5	3.0
1	Bob	92	5.5	5.0
3	David	92	5.5	5.0

```python
df['max_rank']=df['Score'].rank(method='max')
df
```

	Name	Score	average_rank	min_rank	max_rank
4	Eve	79	1.0	1.0	1.0
0	Alice	85	2.0	2.0	2.0
2	Charlie	88	3.5	3.0	4.0

5	Frank	88	3.5	3.0	4.0
1	Bob	92	5.5	5.0	6.0
3	David	92	5.5	5.0	6.0

```
df['first_rank']=df['Score'].rank(method='first')
df
```

	Name	Score	average_rank	min_rank	max_rank	first_rank
4	Eve	79	1.0	1.0	1.0	1.0
0	Alice	85	2.0	2.0	2.0	2.0
2	Charlie	88	3.5	3.0	4.0	3.0
5	Frank	88	3.5	3.0	4.0	4.0
1	Bob	92	5.5	5.0	6.0	5.0
3	David	92	5.5	5.0	6.0	6.0

```
df['dense_rank']=df['Score'].rank(method='dense')
df
```

	Name	Score	average_rank	min_rank	max_rank	first_rank	dense_rank
4	Eve	79	1.0	1.0	1.0	1.0	1.0
0	Alice	85	2.0	2.0	2.0	2.0	2.0
2	Charlie	88	3.5	3.0	4.0	3.0	3.0
5	Frank	88	3.5	3.0	4.0	4.0	3.0
1	Bob	92	5.5	5.0	6.0	5.0	4.0
3	David	92	5.5	5.0	6.0	6.0	4.0

```
df.nsmallest(2,'Score')
```

	Name	Score	average_rank	min_rank	max_rank	first_rank	dense_rank
4	Eve	79	1.0	1.0	1.0	1.0	1.0
0	Alice	85	2.0	2.0	2.0	2.0	2.0

```
df.nlargest(2,'Score')
```

	Name	Score	average_rank	min_rank	max_rank	first_rank	dense_rank
1	Bob	92	5.5	5.0	6.0	5.0	4.0
3	David	92	5.5	5.0	6.0	6.0	4.0

This section focuses on methods to recognize data in DataFrame, such as stacking, unstacking, melting, and creating pivot tables.

```
#stack() converting wide data to long format
#analyzing daily tempratures from multiple cities.
#ex: Imagine you have daily temperature data from multiple cities in a
wide format, but you need it in a long format for easier analysis.

data={
    'Date': ['2025-03-01', '2025-03-02'],
    'New York': [30, 32],
    'Chicago': [20, 22],
}
df=pd.DataFrame(data)
print("wide format:\n",df)
print()
df.set_index('Date',inplace=True)
long_format=df.stack().reset_index()
long_format.columns = ['Date', 'City', 'Temperature']
print("long format:\n",long_format)
print()
```

```
wide format:
          Date  New York  Chicago
0  2025-03-01        30       20
1  2025-03-02        32       22

long format:
          Date      City  Temperature
0  2025-03-01  New York           30
1  2025-03-01   Chicago           20
2  2025-03-02  New York           32
3  2025-03-02   Chicago           22
```

```
#unstack()-converting long data to wide format
#you have long-format data and need to summarize it by city.
wide_format=long_format.set_index(['Date','City']).unstack()
wide_format
```

```
           Temperature
City           Chicago New York
Date
2025-03-01          20       30
2025-03-02          22       32
```

```
#melt()-unpivoting data
#suppose you have a survey dataset where each column represents a
question and each row represents a respone. You want to analyze
answers question-wise.
survey = pd.DataFrame({
```

```
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Q1': [5, 3, 4],
    'Q2': [4, 5, 2]
})

# Unpivoting using melt()
melted = pd.melt(survey, id_vars=['Name'], var_name='Question',
value_name='Rating')
print("\nMelted DataFrame:\n", melted)


Melted DataFrame:
       Name Question  Rating
0    Alice       Q1       5
1      Bob       Q1       3
2  Charlie       Q1       4
3    Alice       Q2       4
4      Bob       Q2       5
5  Charlie       Q2       2

#pivot() - Creating a Pivoted Table
#Imagine you have a sales dataset where each row is a transaction, and
you want to summarize sales by date and product.
sales = pd.DataFrame({
    'Date': ['2025-01-01', '2025-01-01', '2025-01-02'],
    'Product': ['A', 'B', 'A'],
    'Revenue': [100, 200, 150]
})

# Pivoting data to see daily revenue per product
pivoted = sales.pivot(index='Date', columns='Product',
values='Revenue')
print("\nPivoted Sales Data:\n", pivoted)


Pivoted Sales Data:
 Product         A      B
Date
2025-01-01  100.0  200.0
2025-01-02  150.0    NaN

#pivote_table() - creating aggregated pivoted tables
#you have montly sales data and want to summarize revenue and qunatity
sold for each product
data = {
    'Month': ['Jan', 'Jan', 'Feb', 'Feb'],
    'Product': ['A', 'B', 'A', 'B'],
    'Revenue': [1000, 1500, 1100, 1400],
    'Quantity': [50, 60, 55, 58]
}
```

```
df = pd.DataFrame(data)
pivot_table=df.pivot_table(index='Month',
columns='Product',values=['Revenue','Quantity'],aggfunc='sum')
print(pivot_table)
```

```
        Quantity     Revenue
Product        A   B       A     B
Month
Feb           55  58    1100  1400
Jan           50  60    1000  1500
```

```
import pandas as pd

# Example data with mixed date formats
data = {
    'dates': ['2025/03/15', '15-03-2025', '03.15.2025', 'March 15,
2025', '2025-03-15T14:30:00']
}

df = pd.DataFrame(data)

# Convert to a uniform date format (e.g., YYYY-MM-DD)
df['formatted_dates'] = pd.to_datetime(df['dates'],
errors='coerce').dt.strftime('%y-%m-%d')

print("Converted Date Formats:\n", df)
```

```
Converted Date Formats:
                  dates formatted_dates
0           2025/03/15        25-03-15
1           15-03-2025             NaN
2           03.15.2025             NaN
3       March 15, 2025             NaN
4  2025-03-15T14:30:00             NaN
```

```
df = pd.DataFrame({'date': pd.date_range('2023-01-01', periods=4,
freq='D'), 'value': [1, 2, 3, 4]})
print(df['date'].dt.year,
df['date'].dt.month,
df['date'].dt.day)
```

```
0    2023
1    2023
2    2023
3    2023
Name: date, dtype: int32 0    1
1    1
2    1
3    1
```

```
Name: date, dtype: int32 0    1
1    2
2    3
3    4
Name: date, dtype: int32
```

```python
print(df['date'].dt.day_name())
```

```
0       Sunday
1       Monday
2      Tuesday
3    Wednesday
Name: date, dtype: object
```

```python
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'date': pd.date_range('2023-01-01', periods=5,
freq='D'), 'value': [1, 2, 3, 4, 5]})
df['date'] = pd.to_datetime(df['date'])
df.set_index('date', inplace=True)

# Time Series Operations
print("Original:\n", df)
print("\nResample (2D sum):\n", df.resample('2D').sum())
print("\nShift (1 period):\n", df.shift(1))
print("\nRolling (3-day mean):\n", df.rolling(window=3).mean())
print("\nMonth:\n", df.index.month)
print("\nDate range:\n", pd.date_range('2023-01-01', periods=3,
freq='H'))
print("\nAsfreq (12H):\n", df.asfreq('12H', method='ffill'))
df.index = df.index.tz_localize('UTC')
print("\nTZ localize (UTC):\n", df)
df.index = df.index.tz_convert('US/Pacific')
print("\nTZ convert (US/Pacific):\n", df)
```

```
Original:
             value
date
2023-01-01      1
2023-01-02      2
2023-01-03      3
2023-01-04      4
2023-01-05      5

Resample (2D sum):
             value
date
2023-01-01      3
2023-01-03      7
```

```
2023-01-05      5

Shift (1 period):
             value
date
2023-01-01    NaN
2023-01-02    1.0
2023-01-03    2.0
2023-01-04    3.0
2023-01-05    4.0

Rolling (3-day mean):
             value
date
2023-01-01    NaN
2023-01-02    NaN
2023-01-03    2.0
2023-01-04    3.0
2023-01-05    4.0

Month:
 Index([1, 1, 1, 1, 1], dtype='int32', name='date')

Date range:
 DatetimeIndex(['2023-01-01 00:00:00', '2023-01-01 01:00:00',
               '2023-01-01 02:00:00'],
              dtype='datetime64[ns]', freq='H')

Asfreq (12H):
                      value
date
2023-01-01 00:00:00      1
2023-01-01 12:00:00      1
2023-01-02 00:00:00      2
2023-01-02 12:00:00      2
2023-01-03 00:00:00      3
2023-01-03 12:00:00      3
2023-01-04 00:00:00      4
2023-01-04 12:00:00      4
2023-01-05 00:00:00      5

TZ localize (UTC):
                            value
date
2023-01-01 00:00:00+00:00      1
2023-01-02 00:00:00+00:00      2
2023-01-03 00:00:00+00:00      3
2023-01-04 00:00:00+00:00      4
2023-01-05 00:00:00+00:00      5
```

```
TZ convert (US/Pacific):
                            value
date
2022-12-31 16:00:00-08:00      1
2023-01-01 16:00:00-08:00      2
2023-01-02 16:00:00-08:00      3
2023-01-03 16:00:00-08:00      4
2023-01-04 16:00:00-08:00      5
```

```python
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({
    'Name': ['  Alice  ', 'Bob Jones', 'Charlie Brown'],
    'ID': ['X-123', 'Y-456', 'Z-789']
})

# String Operations
print("Cleaned Names:\n", df['Name'].str.strip())
print("\nFirst Names:\n", df['Name'].str.split().str[0])
print("\nUppercase IDs:\n", df['ID'].str.upper())
print("\nLower caseL\n",df['Name'].str.lower())
print("\ncontains:e\n",df['Name'].str.contains('e'))
print("\nExtract Letters:\n", df['ID'].str.extract(r'([A-Z])'))
print("\nReplace Dash:\n", df['ID'].str.replace('-', '_'))
print("\nlength:\n",df['Name'].str.len())
```

```
Cleaned Names:
 0            Alice
1        Bob Jones
2    Charlie Brown
Name: Name, dtype: object

First Names:
 0      Alice
1        Bob
2    Charlie
Name: Name, dtype: object

Uppercase IDs:
 0    X-123
1    Y-456
2    Z-789
Name: ID, dtype: object

Lower caseL
 0          alice
1        bob jones
2    charlie brown
Name: Name, dtype: object
```

```
contains:e
 0    True
1    True
2    True
Name: Name, dtype: bool

Extract Letters:
    0
0  X
1  Y
2  Z

Replace Dash:
 0    X_123
1    Y_456
2    Z_789
Name: ID, dtype: object

length:
 0     9
1     9
2    13
Name: Name, dtype: int64
```

```
import pandas as pd
df = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print(df.add(1))
```

```
   A  B
0  2  4
1  3  5
```

```
df2 = pd.DataFrame({'A': [10, 20], 'B': [30, 40]})
print(df.add(df2))
```

```
    A   B
0  11  33
1  22  44
```

```
print(df.sub(1))
```

```
   A  B
0  0  2
1  1  3
```

```
print(df.mul(2))
```

```
   A  B
0  2  6
1  4  8
```

```
s = pd.Series([2, 3], index=['A', 'B'])
print(df.mul(s))
```

```
   A   B
0  2   9
1  4  12
```

```
print(df.pow(2))
```

```
   A   B
0  1   9
1  4  16
```

```
print(df,"\n")
print(df.sum(axis=0))
print(df.sum(axis=1))
```

```
   A  B
0  1  3
1  2  4

A    3
B    7
dtype: int64
0    4
1    6
dtype: int64
```

```
print(df.mean(axis=1),"\n",df.mean(axis=0))
```

```
0    2.0
1    3.0
dtype: float64
 A    1.5
B    3.5
dtype: float64
```

```
print(df.median())
```

```
A    1.5
B    3.5
dtype: float64
```

```
print(df.std())
```

```
A    0.707107
B    0.707107
dtype: float64
```

```
print(df.var())
```

```
A    0.5
B    0.5
dtype: float64
```

```python
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

# Mathematical Operations
print("Add 10:\n", df.add(10))
print("\nSubtract 1:\n", df.sub(1))
print("\nMultiply by 2:\n", df.mul(2))
print("\nDivide by 2:\n", df.div(2))
print("\nPower of 3:\n", df.pow(3))
print("\nSum:\n", df.sum())
print("\nMean:\n", df.mean())
print("\nMedian:\n", df.median())
print("\nStd Dev:\n", df.std())
print("\nVariance:\n", df.var())
```

```
Add 10:
    A   B
0  11  14
1  12  15
2  13  16

Subtract 1:
   A  B
0  0  3
1  1  4
2  2  5

Multiply by 2:
   A   B
0  2   8
1  4  10
2  6  12

Divide by 2:
     A    B
0  0.5  2.0
1  1.0  2.5
2  1.5  3.0

Power of 3:
    A    B
0   1   64
1   8  125
2  27  216
```

```
Sum:
 A     6
B    15
dtype: int64

Mean:
 A    2.0
B    5.0
dtype: float64

Median:
 A    2.0
B    5.0
dtype: float64

Std Dev:
 A    1.0
B    1.0
dtype: float64

Variance:
 A    1.0
B    1.0
dtype: float64
```

```python
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [4, 3, 6, 5]})

# Statistical Methods
print("Correlation:\n", df.corr())
print("\nCovariance:\n", df.cov())
print("\nSkewness:\n", df.skew())
print("\nKurtosis:\n", df.kurt())
print("\nMean, Median, Mode of A:\n", df['A'].mean(),
df['A'].median(), df['A'].mode())
print("\nMin, Max, Std, Var of B:\n", df['B'].min(), df['B'].max(),
df['B'].std(), df['B'].var())
df['A_cumsum'] = df['A'].cumsum()
df['B_cumprod'] = df['B'].cumprod()
print("\nCumulative Sum and Product:\n", df)
```

```
Correlation:
      A    B
A  1.0  0.6
B  0.6  1.0

Covariance:
```

```
           A         B
A  1.666667  1.000000
B  1.000000  1.666667

Skewness:
 A    0.0
B    0.0
dtype: float64

Kurtosis:
 A   -1.2
B   -1.2
dtype: float64

Mean, Median, Mode of A:
 2.5 2.5 0    1
1    2
2    3
3    4
Name: A, dtype: int64

Min, Max, Std, Var of B:
 3 6 1.2909944487358056 1.6666666666666667

Cumulative Sum and Product:
   A  B  A_cumsum  B_cumprod
0  1  4         1          4
1  2  3         3         12
2  3  6         6         72
3  4  5        10        360
```

```python
import pandas as pd
df = pd.DataFrame({'A': ['a', 'b', 'a']})
df['A'] = df['A'].astype('category')
print(df['A'])
```

```
0    a
1    b
2    a
Name: A, dtype: category
Categories (2, object): ['a', 'b']
```

```python
#df.cat.codes --> Returns the integer codes representing each category
in a categorical column(0-based indexing)
df['A'].cat.codes
```

```
0    0
1    1
2    0
dtype: int8
```

```python
df['A'].cat.categories #Returns the list of categories defined for a categorical column.
```

```
Index(['a', 'b'], dtype='object')
```

```python
df = pd.DataFrame({'A': ['x', 'y', 'x']})
df['A'] = df['A'].astype('category')
df['A'] = df['A'].cat.rename_categories(['cat1', 'cat2'])
print(df)
```

```
      A
0  cat1
1  cat2
2  cat1
```

```python
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'A': ['low', 'medium', 'high', 'low'], 'B': [1, 2, 3, 4]})

# Categorical Data Operations
df['A'] = df['A'].astype('category')
print("As Categorical:\n", df)
print("\nCategory Codes:\n", df['A'].cat.codes)
print("\nCategories:\n", df['A'].cat.categories)
df['A'] = df['A'].cat.rename_categories(['L', 'M', 'H'])
print("\nRenamed Categories:\n", df)
```

```
As Categorical:
         A  B
0     low  1
1  medium  2
2    high  3
3     low  4

Category Codes:
 0    1
1    2
2    0
3    1
dtype: int8

Categories:
 Index(['high', 'low', 'medium'], dtype='object')

Renamed Categories:
    A  B
0  M  1
1  H  2
```

```
2  L  3
3  M  4

df1 = pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
df2 = pd.DataFrame({'A': [2, 3], 'B': [5, 6]})
print(pd.concat([df1,df2],axis=0))

   A  B
0  1  3
1  2  4
0  2  5
1  3  6

df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}, index=['x', 'y',
'z'])
df2 = pd.DataFrame({'A': [2, 3, 4], 'B': [5, 6, 7]}, index=['y', 'z',
'w'])
df_intersection = df1.merge(df2, on=['A', 'B'], how='inner')
print(df_intersection)

   A  B
0  2  5
1  3  6

df_diff = df1.loc[df1.index.difference(df2.index)]  # Replacing
df1.difference(df2)
print(df_diff)

   A  B
x  1  4

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
print(df['A'].isin([1, 2]))

0     True
1     True
2    False
Name: A, dtype: bool

import pandas as pd

# Create DataFrames
df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]}, index=['x', 'y',
'z'])
df2 = pd.DataFrame({'A': [2, 3, 4], 'B': [5, 6, 7]}, index=['y', 'z',
'w'])

# Set Operations
df_union = pd.concat([df1, df2])
print("Union:\n", df_union)
df_intersection = df1.loc[df1.index.intersection(df2.index)]
```

```
print("\nIntersection:\n", df_intersection)
df_diff = df1.loc[df1.index.difference(df2.index)]
print("\nDifference:\n", df_diff)
print("\nIsin [2, 3] for A:\n", df1['A'].isin([2, 3]))

Union:
    A  B
x  1  4
y  2  5
z  3  6
y  2  5
z  3  6
w  4  7

Intersection:
    A  B
y  2  5
z  3  6

Difference:
    A  B
x  1  4

Isin [2, 3] for A:
 x    False
y     True
z     True
Name: A, dtype: bool

import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3, 4], 'B': [5, 4, 3, 2]})
print(df.query('A > 2 and B < 5'))

   A  B
2  3  3
3  4  2

threshold = 3
print(df.query('A > @threshold'))
print(df.query('A % 2 == 0 and B >= 4'))

   A  B
3  4  2
   A  B
1  2  4

df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df.eval('C = A + B', inplace=True)
print(df)
```

```
   A  B  C
0  1  4  5
1  2  5  7
2  3  6  9
```

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': ['x', 'y', 'z']})
print(df.memory_usage(deep=True))
```

```
Index    132
A         24
B        174
dtype: int64
```

```
import pandas as pd

# Create a DataFrame
df = pd.DataFrame({'A': [1, 2, 3, 4, 5], 'B': [6, 5, 4, 3, 2]})

# Performance Optimization
print("Query A > 3 and B < 4:\n", df.query('A > 3 and B < 4'))
df.eval('C = A * B', inplace=True)
print("\nEval C = A * B:\n", df)
print("\nMemory Usage (deep=True):\n", df.memory_usage(deep=True))
```

```
Query A > 3 and B < 4:
    A  B
3  4  3
4  5  2

Eval C = A * B:
    A  B   C
0  1  6   6
1  2  5  10
2  3  4  12
3  4  3  12
4  5  2  10

Memory Usage (deep=True):
 Index    132
A         40
B         40
C         40
dtype: int64
```

used to prefom calculations on a set of data points (referred to as a 'window') defined by the user. these functions help in analyzind data over a specific range of rows or groups without affecting the overall dataset.

```python
import pandas as pd
df = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6]})
df.rolling(window=2).mean()
```

	A
0	NaN
1	1.5
2	2.5
3	3.5
4	4.5
5	5.5

```python
df.rolling(window=4,min_periods=1).sum()
#min_periods=1 specifies the minimum number of observations required
within the window to compute a result.
```

	A
0	1.0
1	3.0
2	6.0
3	10.0
4	14.0
5	18.0

```python
#df.expanding() where the window grows from the start of the data to
the current row, useful for cumulative statistics.
df['expanding_sum']=df['A'].expanding().sum()
df['expanding_mean']=df['A'].expanding().mean()
df
```

	A	expanding_sum	expanding_mean
0	1	1.0	1.0
1	2	3.0	1.5
2	3	6.0	2.0
3	4	10.0	2.5
4	5	15.0	3.0
5	6	21.0	3.5

```python
import pandas as pd

df = pd.DataFrame({'A': [1, 2, 3, 4, 5]})
df['ewm'] = df['A'].ewm(span=10).mean()
print(df)
#The exponentially weighted average gives more importance to recent
data points by using a higher weight (α) for newer values and less
weight (1 - α) for older ones.
```

	A	ewm
0	1	1.000000
1	2	1.550000
2	3	2.132890

```
3  4  2.748020
4  5  3.394502
```

```python
#df.apply() applies a function along an axis (rows or columns) of the DataFrame.
import pandas as pd
df=pd.DataFrame({'A': [1, 2], 'B': [3, 4]})
print(df.apply(sum,axis=0))
```

```
A    3
B    7
dtype: int64
```

```python
def double(x):
    return x * 2
print(df.apply(double))
```

```
   A  B
0  2  6
1  4  8
```

```python
#df.applymap() applies function element-wise to every value in data frame
print(df.apply(lambda x:x*2))
df.applymap(lambda x:x*2)
```

```
   A  B
0  2  6
1  4  8
```

```
   A  B
0  2  6
1  4  8
```

```python
import pandas as pd

# Create a DataFrame and Series
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
s = pd.Series(['x', 'y', 'z'])

# Applying Functions
print("Apply sum (columns):\n", df.apply(sum))
print("\nApplymap double:\n", df.applymap(lambda x: x * 2))
print("\nPipe add 3:\n", df.pipe(lambda df: df + 3))
print("\nMap rename:\n", s.map({'x': 'X', 'y': 'Y', 'z': 'Z'}))
print("\nTransform triple:\n", df['A'].transform(lambda x: x * 3))
```

```
Apply sum (columns):
 A     6
B    15
dtype: int64
```

```
Applymap double:
   A   B
0  2   8
1  4  10
2  6  12

Pipe add 3:
   A  B
0  4  7
1  5  8
2  6  9

Map rename:
 0    X
1    Y
2    Z
dtype: object

Transform triple:
 0    3
1    6
2    9
Name: A, dtype: int64
```