Pandas

- Install as : pip install pandas
- Used for data manipulation & visualization
- Built on top of NumPy & Matplotlib
- 2-D labelled data structures
- Different columns are potentially different data types
- We can manipulate, slice, reshape, groupby, join & merge data
- We can also know statistics and handle missing values
- Easier to work with time series

Read CSV

- For reading csv file
- Example:

```
import pandas as pd
df = pd.read_csv(file_path)   # Load CSV file
print(df.head())        # Display Top 5 rows
print(df.tail(3))       # Display bottom 3 rows
np_array = df.values    # Convert data into NumPy array (Check shape, type)
print(df.columns)       # Display header of data
```

Other Args

- `sep = '\t'` ➔ Separator of fields. Here, its tab
- `quotechar = '"'` ➔ Fields enclosed inside given symbol
- `nrows = 4` ➔ Read only first four row (except header)
- `skiprows = 2` ➔ Skip first 2 row (including header)
- `header = None` ➔ Without header (2 if header on 2nd row)
- `names = lis_h` ➔ Set header name from given list
- `usecols = lis_c` ➔ Load given columns (list of index/header)
- `index_col = lis_i` ➔ Column that is to be used as index
- `parse_date = True` ➔ Load potential date / [1] if 1st col is date

Load Excel

```
# pip install openpyxl
sheet_df = pd.read_excel(file)      # loads first_sheet
sheet_df = pd.read_excel(file, sheet_name=None)  # Load all sheet
```

From Database

```
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine(connection_string)
df = pd.read_sql_query(SQL, engine)
engine.dispose()
```

Connection Strings

- Sqlite: 'sqlite:///db_name.sqlite'
- Postgres: 'postgresql+psycopg2://db_user:db_password@db_host:db_port/db_name'
- MySQL: 'mysql+pymysql://db_user:db_password@db_host:db_port/db_name'

Pandas Series

- 1-D data structure with header and value
- NumPy operation is applicable. Operation is element wise

```
df = pd.read_csv(basketball_player)
age_series = df['age']
age_series[age_series < 20]
bmi_series = df['weight'] / df['height'] ** 2
print(age_series + 20)
print(age_series.values)   # Convert it to NumPy array
```

Creating data

- We may need to create data from existing list or array

```
import pandas as pd

cities = ['kathmandu', 'pokhara', 'biratnagar']
visitors = [2, 3, 7]
temp = [25, 27, 35]
data_dict = {'city': cities, 'visitors': visitors, 'temp': temp}

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

Exercise

EDA

- Head
- Tail
- Sample
- Info
- Describe
- Dtype

Write Data Frame

```
df.to_csv('final_data.csv’)
```

- We can choose separation, headers to be present or not, index to be present or not, quotechar during exporting file to csv

```
df.to_csv('final_data.csv', sep='\t', index=False, header=False)
df.to_csv('final_data.csv', quotechar='"', index=False)
df.to_excel('output_filename’)
df.to_sql('table_name', engine, if_exists='replace', index=False)
# if_exists: 'fail', 'replace', 'append'
```

Broadcasting

```
import pandas as pd

name = ['Ram', 'Shyam', 'Sita']
age = [23, 30, 27]
wt = [58, 62, 60]
ht = [1.25, 2.7, 1.35]
data_dict = {'name':name, 'weight':wt, 'height':ht, 'country':'Nep'}
data_df = pd.DataFrame(data_dict)

# Broadcasting
data_df['year'] = 2025
data_df['bmi'] = data_df.weight / data_df.height ** 2
data_df['age'] = age
print(data_df.head())
data_df['age'] = data_df['age'] + 5
```

Sorting

- Data can be sorted based on a column or combination

```
df = data_df.sort_values(['age'], ascending=[False])
df = data_df.sort_values(['age', 'bmi'], ascending=[True, False])
print(df.head())

df_index_sorted = df.sort_index(ascending=False)
print(df_index_sorted.head())
```

Index Operation

```
nid_data = ['123', '121AC', '23DC']
df['nid'] = nid_data
df = df.set_index('nid')  # df.index=nid_data, df.index.name='nid'
print(df.head())
df = df.reset_index() # drop=True
```

Dropping Data

- We can omit data from row or column
- Axis argument is used to defined whether data has to be removed from row or column

```
data_df = data_df.drop(['123'])     # Drop row based on index
data_df = data_df.drop(['age'], axis=1)  # Drop age column in df
```

Rename Column

```
my_df = my_df.rename(columns={'ExistingName': 'NewName'})
```

- We can provide multiple key-value pairs to rename multiple columns

Change Data Type

```
data_df['age'] = data_df.age.astype(str)
print(data_df.age + '1')
```

Select

- Also called sub-netting

Examples:

```
df['age'][0:2]      # 1st and 2nd row of age column
df.loc[:2, 'age']   # Same as above. Here its row first, inclusive
df.iloc[:2, :2]     # index loc. Based on position. Exclusive

df['age']
df[['age']]
df[['age', 'weight']]
df.loc['KTM':'PKR', :]   # Row index from KTM to PKR but all values
df.loc['KTM':'PKR', 'Age':'Weight']      # Row & Column indexing
df.loc['KTM':'PKR', ['age', 'weight']]   # Only two column in slice
df.ioc[[0,1], 1:]
```

Filter

- Similar to NumPy filter
- We create a mask and filter with help of it
- We can use relational operator (<,>, <=, >=, ==, !=) for mask condition
- df.col_num.isin(chk_list) # for checking in condition

```
mask_1 = df.age > 20
df[mask_1] # or df[df.age > 20]
df[(mask_1) & (mask_2)]
df[(mask_1) | (mask_2)]
df[~mask_1]
```

Group By

- For aggregating data
- **Syntax**: df.groupby([columns])[[other_columns]].sum() # as_index=False
- We can use user-defined function as well in .agg (np.min)

```
df.groupby(['a', 'b'], as_index=False)[['c', 'd']].sum()
df.groupby(['a'])[['c', 'd']].agg({'a':'sum', 'b': 'mean')
df.groupby(['a'])[['c', 'd']].agg(['max', 'min'])
df.column.mean()  # df.column.agg(use_defined_function)
df.groupby(['a'].agg(new_col_name=(old_col, agg_function), ..)
```

Aggregators

- df.column.aggregators() / df[['col_1', 'col_2']].aggregators() / df.aggre()
- Some aggregators:
- **.count**, **.mean**(axis=1), **.std**(), **.median**(), **.quantile**(q), **.min**(), **.idxmin**(), **.max**(), **.idmax**(), **.describe**(), **.unique**(), **.nunique**(), **.sum**(), **.cumsum**(), **.prod**(), **cumprod**(), **.cummax**(), **.mode**(), **.var**()
- df = pd.DataFrame({'ht': [2, 4, 1, 2], 'wt': [1, 3, 5, 1]})

Merge

- Similar to join in SQL

```
merged_df = pd.merge(df_1, df_2, on='id')
merged_df = df_1.merge(df_2, on='id', how='left')
```

Other Arguments:

- Left ➔ Left Data Frame Name
- Right ➔ Right Data Frame Name
- on ➔ Join Condition
- left_on ➔ Join Condition of Left Table
- right_on ➔ Join Condition on Right Table
- suffixes ➔ Suffixes to add to column names of table
- how ➔ Specify Type of Join. Inner, left, right, outer

Concat

- Generally huge data is stored in multiple files or we may date new file for data of each day. In such we need to combine them and evaluate
- We can combine data column wise if index (similar to join)

```
import pandas as pd

df_1 = pd.DataFrame({‘CON’:[‘NEP’, ‘CHN’], ‘CAP’: [‘KTM’, ‘Beijing’]})
df_2 = pd.DataFrame({‘CON’:‘IND’, ‘CAP’: ‘Delhi’}, index=‘0’)
df_3 = pd.DataFrame({‘CON’:‘BAN’, ‘CAP’:‘Dhaka’}, index=‘0’)
merged_data = pd.concat([df_1, df_2, df_3])   # ignore_index=True
```

Time Series

- Analysis based on time. Series with index as datetime

```
import pandas as pd

df_1 = pd.read_csv('daily_minimum_temp', parse_dates=['Date'], index_col='Date')
df.loc['1981':'1982']     # or .loc['1981']
df.loc[['1981', '1982']]  # Doesn't give expected result
df.resample('ME').mean()  # Monthly mean
df.resample('YE').sum()       # Yearly Max
df.resample('M').mean().max()
```

Generic Function

- We can access the function of a data type using its extension at last
- Example:
  - df['name'] = df.name.str.lower() # Note .str and lower is its func
  - df['Year'] = df.date.dt.year # Note .dt and year is its func
  - print(df[df.name.str.contains('ab')])

- Que: Find the longest word in data frame
- Que: Data Frame has age,gender in single column. Store it into two

Exercise