READ FILES:
pd.read_csv()
pd.read_json()
pd.read_fwf()
df.to_csv()

```
from google.colab import drive
drive.mount('/content/drive')

hw=pd.read_csv("/content/drive/My Drive/DataManagement/data_files/weight-height.csv")
```

Series:
pd.Series(data =, index =
pd.Series(dict)

```
gdrivePath="/content/drive/My Drive/DataManagement/data_files/"
import sqlite3
import pandas as pd
import os
import pandasql
```

DataFrames:
pd.DataFrame(data = , index =, columns =)
pd.DataFrame(dict)

eg.	rnums=pd.DataFrame(np.random.randint(1,11,size=100).reshape(10,10))

Display:
df.head(n) #show first n rows, default 5 rows
df.describe() #show description

Drop:
df.drop(labels, axis =, inplace = True/False) #axis = 1: column; axis = 0: index

```
c.drop('fy',axis=1,inplace=True)
```

del df[columns] `del c['mcap']`

Indexing:
df.loc(index, columns)
df.iloc(index of index, index of columns)

pd.NA, np.nan:
df.isnull() # Series.isnull()
df.notnull() #Series.notnull()
df.dropna(axis=) # drop rows/columns where at least one element is missing
df.fillna(content) or df.fillna(methods = 'ffill')

```
df.fillna({'c':df['c'].mean(),'d':df['d'].mean()})
```

Vectorized Str:
c['location'].str.upper()
c['revenue'].str[1:].astype(float)

```
tmp = df['URL'].str.split('://', expand=True)
```

	0	1
0	https	nyu.edu
1	http	apple.com

```
tmp = df['URL'].str.split('://', expand=False)
```

```
0     [https, nyu.edu]
1    [http, apple.com]
Name: URL, dtype: object
```

SQL:

1. **conn** = **sqlite3.connect**(gdrivePath+'employee.db')
2. **pd.read_sql_query**(query, conn, index_col =)

```
def execSQL(conn,query):
  conn.execute(query) # execute an SQL query
  conn.commit() # "commit" that query in order to make its action permanent
def allrowsSelect(conn,query):
  cursor = conn.execute(query)
  for row in cursor:
    print(row)
```

3.CREATE TABLE

```
query = '''
CREATE TABLE employee(
      id integer PRIMARY KEY,
      name text,
      salary real
);
'''
execSQL(conn,query)
```

4. **INSERT**

```
query = '''
INSERT INTO employee (id, name, salary)
      VALUES
            (1, ' A', 100.0),
            (2, 'B', 105.5),
            (3, 'C', 110.7);
'''
execSQL(conn, query)
```

5. **SELCT**

```
query = '''
SELECT (DISTINCT) salary, count() AS num
FROM employee
WHERE last_name LIKE 'K_K%'
GROUP BY salary
HAVING num > 100
ORDER BY num
LIMIT 1 OFFSET 2
```

6. **ALTER (ADD,DRO, RENAME)**

```
ALTER TABLE table_name
ADD COLUMN col_name data_type
DROP COLUMN col_name
RENAME COLUMN old_name TO new
```

7. **UPDATE**

```
query = ' UPDATE employee SET genre = 'Fiction' WHERE type = 'science'
```

8. **DELETE:** query = ' **DELETE FROM** employee **WHERE** lastname = 'Brown' '
9. **DROP:** query = '**DROP TABLE IF EXISTS** table_name'

Rearranging:
df.reindex(index = [], columns =[])

```
c=c.reindex(index=[*(list(c.index)[1:]), 'apple', 'microsoft'],
        columns=['city','state','country','revenue', 'employees'])
```

df.rename(index, columns)

```
df2=df.rename(columns=str.upper)
df2=df.rename(columns={c: c.upper() for c in list(df.columns)})
df.columns=df.columns.map(str.upper)
df.columns=['X','Y','Z']
df.index=[1,2,3]
```

Replace:
pd.replace(old, new)

```
df[1]=df[1].str.upper().str.replace('.','!',regex=False)
```

Apply and Map
df.apply(func, axis =) : 0 to column, 1 to row
Series.apply(func)

Series.map(func): all element
df.map(func): all element

Sorting
df.sort_values(by=, ascending=)
eg. c.sort_values(by='rev_emp',ascending=False)
Series.unique() # give unique value in series
df.value_counts() #Series.value_counts() #give value and its occurrence frequency

Binning Values (CUT)
pd.cut(x, bins =, labels=)

```
bins = [0,100,130,160,190, float('inf')]
w = pd.cut(p3[p3['Gender'] == 'Female']['Weight'], bins=bins, labels=['less than 100', '100-130', '130-160', '160-190', 'over 190'])
w.value_counts()
```

Type Conversion:
pd.to_numeric(x, errors=) #convert columns to numeric, errors='coerce' make error to NA
pd.to_datetime(x) #convert str in Series to datetime

Merge and Concat
pd.merge(a,b,on=, how=) #how: left, right, inner, outer, cross; drop or add corresponding columns
pd.concat([a,b], axis =) #add rows/columns of second df at the end of the first df

Set Index and Group by:
df.set_index(columns, drop=True) #set target columns to the index, drop the original index in the df
df.reset_index() #set the index back to columns, and use default index
df.groupby(columns).func() #set the columns to index, and apply func to the groups based on the columns

```
df[['Title','Employment', 'Salary']].groupby('Title').mean()
df[['Employment', 'Salary']].groupby(df['Title']).mean()
```

Aggregation:

```
df.groupby('State').aggregate({'Salary': func, 'Employment': lambda x: np.std(x)})
```

State	Salary	Employment
CA	90640.000000	4839.306654
NY	88071.666667	2416.234444

11. **GROUP_CONCAT()**
12. **SUBSTRING(column, where to start(count start 1), how many bit)**
13. **conn.execute("CREATE TABLE leaders AS" + query)**

```
pd.read_sql_query("select name from sqlite_master where type='table' or type='view'", conn5)
print(pd.read_sql_query("SELECT * FROM pragma_table_info(table)", conn5)
```

```
if os.path.exists('employee.db'):
  os.remove('employee.db')
conn = sqlite3.connect('employee.db')
```

```
customer=pd.merge(sales_df,customers_df,on='CUSTOMER_ID',how='right')[['CUSTOMER_ID',
'Customer', 'SALE_ID']]
customer[customer['SALE_ID'].isnull()][['CUSTOMER_ID','Customer']]
```

7. **UPDATE**

query = ' **UPDATE** employee **SET** genre = 'Fiction' **WHERE** type = 'science'

8. **DELETE:** query = ' **DELETE FROM** employee **WHERE** lastname = 'Brown' '
9. **DROP:** query = '**DROP TABLE IF EXISTS** table_name'
9. **to_sql**

```
oil_imports.to_sql('oil_imports',conn2)
```

```
customer=pd.merge(sales_df,customers_df,on='CUSTOMER_ID',how='right')[['CUSTOMER_ID',
'Customer', 'SALE_ID']]
customer[customer['SALE_ID'].isnull()][['CUSTOMER_ID','Customer']]
```