Python Pandas Tutorials (CWH)

Video Link: https://youtu.be/RhEjmHeDNoA (https://youtu.be/RhEjmHeDNoA)

In [1]:

```
import numpy as np
import pandas as pd
```

Pandas is a open source data analysis library written in python. It leverages the power and speed of `numpy` to make data analysis and preprocessing easy. It's a tool like **excel**.

In [2]:

```
dict1 = {
    'paper': ['CC08', 'CC09', 'CC10', 'GE2', 'SEC2'],
    'topics': ['Mathematical Physics', 'Modern Physics',
               'Analog Electronics', 'Chemistry', 'Radiation Safety'],
    'credits': [4, 4, 4, 2, 2],
    'faculties': ['DB, AG', 'DB, CM', 'SC, AP', np.nan, 'AP, CM, SC'],
    'lectures': ['NPTEL, GP, Vishwakarma', 'HCV', 'HCV, Neso',
                  'Harshita Khurana', np.nan],
    'books': ['BS Grewal', 'SN Ghoshal', 'Boylestad, B Ghosh',
               'Santra', np.nan]
}
df1 = pd.DataFrame(dict1)
display(df1)

df1.to_csv('sem 4 overview.csv', index=False) # creating a csv file
```

	paper	topics	credits	faculties	lectures	books
0	CC08	Mathematical Physics	4	DB, AG	NPTEL, GP, Vishwakarma	BS Grewal
1	CC09	Modern Physics	4	DB, CM	HCV	SN Ghoshal
2	CC10	Analog Electronics	4	SC, AP	HCV, Neso	Boylestad, B Ghosh
3	GE2	Chemistry	2	NaN	Harshita Khurana	Santra
4	SEC2	Radiation Safety	2	AP, CM, SC	NaN	NaN

In [3]:
```
display(df1.head(3), df1.tail(3))
display(df1.describe())
```

	paper	topics	credits	faculties	lectures	books
0	CC08	Mathematical Physics	4	DB, AG	NPTEL, GP, Vishwakarma	BS Grewal
1	CC09	Modern Physics	4	DB, CM	HCV	SN Ghoshal
2	CC10	Analog Electronics	4	SC, AP	HCV, Neso	Boylestad, B Ghosh

	paper	topics	credits	faculties	lectures	books
2	CC10	Analog Electronics	4	SC, AP	HCV, Neso	Boylestad, B Ghosh
3	GE2	Chemistry	2	NaN	Harshita Khurana	Santra
4	SEC2	Radiation Safety	2	AP, CM, SC	NaN	NaN

	credits
count	5.000000
mean	3.200000
std	1.095445
min	2.000000
25%	2.000000
50%	4.000000
75%	4.000000
max	4.000000

Working with `Csv` files

In [4]:
```
sem4 = pd.read_csv("sem 4 overview.csv")
display(sem4['topics'])
print(type(sem4), '\t', type(sem4['topics']))
```

```
0    Mathematical Physics
1          Modern Physics
2      Analog Electronics
3               Chemistry
4        Radiation Safety
Name: topics, dtype: object
```

```
<class 'pandas.core.frame.DataFrame'>    <class 'pandas.core.series.Series'>
```

In [5]:

```
cc06prac3 = pd.read_csv('single slit diffraction data.csv')
cc06prac3.index = [5,4,3,2,1,0,1,2,3,4,5]
display(cc06prac3)
#cc06prac3['mean reading'][0] = 3.15 # changing a data
LtoR = cc06prac3['L to R'].to_numpy()
RtoL = cc06prac3['R to L'].to_numpy()
mean_readings = (LtoR + RtoL)/2
display('left to right readings', LtoR,
        'right to left readings', RtoL,
        'mean readings', mean_readings)
```

	order	L to R	R to L	mean reading
5	5th	3.2	3.2	3.20
4	4th	2.8	2.8	2.80
3	3rd	2.5	2.4	2.45
2	2nd	2.2	2.0	2.10
1	1st	1.7	1.5	1.60
0	central	1.2	1.0	1.10
1	1st	0.4	0.2	0.30
2	2nd	-0.2	-0.2	-0.20
3	3rd	-0.8	-0.7	-0.75
4	4th	-1.1	-1.2	-1.15
5	5th	-1.7	-1.6	-1.65

```
'left to right readings'

array([ 3.2,  2.8,  2.5,  2.2,  1.7,  1.2,  0.4, -0.2, -0.8, -1.1, -1.7])

'right to left readings'

array([ 3.2,  2.8,  2.4,  2. ,  1.5,  1. ,  0.2, -0.2, -0.7, -1.2, -1.6])

'mean readings'

array([ 3.2 ,  2.8 ,  2.45,  2.1 ,  1.6 ,  1.1 ,  0.3 , -0.2 , -0.75,
       -1.15, -1.65])
```

Data Structure:

Pandas has 2 types of data structures:

1. **Series:** It's a one dimensional array with indexes, it stores a singel column or row of data in a *dataframe*. It's capable of holding any one type of data.
2. **Dataframe:** It's a tabular spreadsheet like structure responding rows each of which contains one or multiple columns. It's a 2 dimensional structure with columns of potentially different types of data.

In [6]:

```
ser1 = pd.Series(np.random.rand(20))
display(type(ser1), ser1.head(7))
```

```
pandas.core.series.Series
```

```
0    0.643929
1    0.601468
2    0.929544
3    0.640078
4    0.498413
5    0.715607
6    0.645700
dtype: float64
```

In [7]:

```
newdf1 = pd.DataFrame(np.random.randn(110, 6))
display(type(newdf1), newdf1.head(), newdf1.dtypes)
display(newdf1.describe())
newdf1[0][0] = 'random data'
display(newdf1.head(4), newdf1.dtypes)
```

```
pandas.core.frame.DataFrame
```

	0	1	2	3	4	5
0	0.519252	-0.107567	0.069394	-1.845216	0.551087	-0.009382
1	-0.934733	-0.530735	0.471899	-0.415554	-0.557208	-1.476375
2	0.776062	-1.549687	0.880336	1.261908	-0.205181	1.775464
3	-0.805187	0.619220	1.123326	0.273783	0.217997	-2.427237
4	0.873255	0.200943	0.247609	0.978975	-1.092949	0.739092

```
0    float64
1    float64
2    float64
3    float64
4    float64
5    float64
dtype: object
```

	0	1	2	3	4	5
count	110.000000	110.000000	110.000000	110.000000	110.000000	110.000000
mean	0.002536	-0.038344	-0.091609	0.051691	-0.043163	0.134386
std	0.998856	0.977671	1.108484	0.981535	1.024114	1.114257
min	-3.011975	-2.191568	-2.530668	-2.099707	-2.259446	-2.776113
25%	-0.742359	-0.779756	-0.993726	-0.581155	-0.663172	-0.561860
50%	0.096946	-0.044301	0.032851	0.036068	-0.050255	0.247565
75%	0.786921	0.483731	0.790635	0.739346	0.627697	0.822252
max	3.477013	2.691216	2.287651	2.739050	3.363496	3.073639

	0	1	2	3	4	5
0	random data	-0.107567	0.069394	-1.845216	0.551087	-0.009382
1	-0.934733	-0.530735	0.471899	-0.415554	-0.557208	-1.476375
2	0.776062	-1.549687	0.880336	1.261908	-0.205181	1.775464
3	-0.805187	0.619220	1.123326	0.273783	0.217997	-2.427237

```
0     object
1    float64
2    float64
3    float64
4    float64
5    float64
dtype: object
```

index, columns, to_numpy, info and others

In [8]:
```
display(newdf1.index)
display(newdf1.columns)
newdf1[0][0] = np.pi
display(newdf1.T)  # transpose
display(newdf1.to_numpy())
```

```
RangeIndex(start=0, stop=110, step=1)

RangeIndex(start=0, stop=6, step=1)
```

	0	1	2	3	4	5	6	7	8	
0	3.141593	-0.934733	0.776062	-0.805187	0.873255	0.973343	0.562907	-0.102692	0.129879	-(
1	-0.107567	-0.530735	-1.549687	0.61922	0.200943	1.575089	0.41145	-1.131892	-0.733053	-(
2	0.069394	0.471899	0.880336	1.123326	0.247609	-0.047594	-0.516616	-1.979243	0.269876	
3	-1.845216	-0.415554	1.261908	0.273783	0.978975	-0.343533	-2.03535	-0.214226	-0.582842	(
4	0.551087	-0.557208	-0.205181	0.217997	-1.092949	0.578384	0.620194	0.101504	-0.144189	1
5	-0.009382	-1.476375	1.775464	-2.427237	0.739092	0.840836	-0.135106	-1.06254	-0.840113	-2

6 rows × 110 columns

In [9]:
```
display(newdf1.sort_index(axis=0, ascending=False))
# axis=0 for rows and axis=1 for columns
display(newdf1[1].head(4), type(newdf1[1]))
```

	0	1	2	3	4	5
109	0.796564	0.322604	-0.177676	0.580895	1.148047	3.073639
108	0.861966	0.062173	-1.128290	-1.280898	1.364801	0.766037
107	-1.605694	-0.789346	-0.313562	0.207263	1.025265	2.059027
106	0.824008	-1.268946	-0.265850	0.596431	-1.880360	0.033423
105	-0.198477	1.091105	-1.489506	-0.438914	0.583630	0.755152
...	...	...	...	...	...	...
4	0.873255	0.200943	0.247609	0.978975	-1.092949	0.739092
3	-0.805187	0.619220	1.123326	0.273783	0.217997	-2.427237
2	0.776062	-1.549687	0.880336	1.261908	-0.205181	1.775464
1	-0.934733	-0.530735	0.471899	-0.415554	-0.557208	-1.476375
0	3.141593	-0.107567	0.069394	-1.845216	0.551087	-0.009382

110 rows × 6 columns

```
0   -0.107567
1   -0.530735
2   -1.549687
3    0.619220
Name: 1, dtype: float64

pandas.core.series.Series
```

In [10]:

```
print(newdf1.shape)
display(newdf1.info())
display(newdf1[0].value_counts(dropna=False).head())
display(newdf1.notnull().head())
```

```
(110, 6)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110 entries, 0 to 109
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       110 non-null    object 
 1   1       110 non-null    float64
 2   2       110 non-null    float64
 3   3       110 non-null    float64
 4   4       110 non-null    float64
 5   5       110 non-null    float64
dtypes: float64(5), object(1)
memory usage: 5.3+ KB

None

 3.141593    1
-0.740844    1
-0.825665    1
-1.071444    1
 0.482282    1
Name: 0, dtype: int64
```

	0	1	2	3	4	5
0	True	True	True	True	True	True
1	True	True	True	True	True	True
2	True	True	True	True	True	True
3	True	True	True	True	True	True
4	True	True	True	True	True	True

copying a dataframe

In [11]:

```
newdf1v = newdf1
newdf1v[0][1] = 9.3
print(newdf1[0][1])
# newdf2 is not a new dataframe, it's just a view of dataframe newdf1
newdf1c = newdf1.copy()  # or newdf1[:]
newdf1c[0][0] = 10
print(newdf1[0][0])
```

```
9.3
3.141592653589793

C:\Users\suman\AppData\Local\Temp\ipykernel_2588\2736990107.py:6: SettingWithCopyWa
rning:
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 (https://pandas.pydata.org/
pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  newdf1c[0][0] = 10
```

`loc` : To have no warnings in output we need to use the function `loc` . It is used to change values of a dataframe.

In [12]:

```
newdf1.loc[1,1] = 5.8 # write [a,b] instead of [a][b]
display(newdf1.head(3))
```

	0	1	2	3	4	5
0	3.141593	-0.107567	0.069394	-1.845216	0.551087	-0.009382
1	9.3	5.800000	0.471899	-0.415554	-0.557208	-1.476375
2	0.776062	-1.549687	0.880336	1.261908	-0.205181	1.775464

In [13]:

```
newdf1c.columns = list('ABCDEF')
display(newdf1c.head())
display(newdf1.drop(0).drop(0, axis=1).head())
```

	A	B	C	D	E	F
0	10	-0.107567	0.069394	-1.845216	0.551087	-0.009382
1	9.3	-0.530735	0.471899	-0.415554	-0.557208	-1.476375
2	0.776062	-1.549687	0.880336	1.261908	-0.205181	1.775464
3	-0.805187	0.619220	1.123326	0.273783	0.217997	-2.427237
4	0.873255	0.200943	0.247609	0.978975	-1.092949	0.739092

	1	2	3	4	5
1	5.800000	0.471899	-0.415554	-0.557208	-1.476375
2	-1.549687	0.880336	1.261908	-0.205181	1.775464
3	0.619220	1.123326	0.273783	0.217997	-2.427237
4	0.200943	0.247609	0.978975	-1.092949	0.739092
5	1.575089	-0.047594	-0.343533	0.578384	0.840836

In [14]:

```
display(newdf1c.loc[[1,2,3],['B','C']])
display(newdf1c.loc[1:5,['A','C']])
display(newdf1c.loc[(newdf1c['A']<0) & (newdf1c['C']>0)].head())
```

	B	C
1	-0.530735	0.471899
2	-1.549687	0.880336
3	0.619220	1.123326

	A	C
1	9.3	0.471899
2	0.776062	0.880336
3	-0.805187	1.123326
4	0.873255	0.247609
5	0.973343	-0.047594

	A	B	C	D	E	F
3	-0.805187	0.619220	1.123326	0.273783	0.217997	-2.427237
9	-0.141258	-0.230773	0.479890	0.918095	1.179069	-2.323365
11	-0.874963	-0.200762	0.758108	-1.768436	-1.738117	0.450379
15	-0.227045	0.786087	0.176291	-0.672259	-0.039814	0.749716
23	-1.597969	-0.097059	0.417908	0.087731	-1.041021	-2.776113

`iloc` : To get values at a particular location by giving index.

In [15]:

```
display(newdf1c.iloc[0,3])
display(newdf1c.iloc[:5, [3,4]])
```

```
-1.8452163641113155
```

	D	E
0	-1.845216	0.551087
1	-0.415554	-0.557208
2	1.261908	-0.205181
3	0.273783	0.217997
4	0.978975	-1.092949

`drop` :

In [16]:

```
display(newdf1c.drop(['E', 'F'], axis=1).head(3))
display(newdf1c.head(3)) # not changed
newdf1c.drop(['E','F'], axis=1, inplace=True)
display(newdf1c.head(3)) # changed (when inplace is used)
```

	A	B	C	D
0	10	-0.107567	0.069394	-1.845216
1	9.3	-0.530735	0.471899	-0.415554
2	0.776062	-1.549687	0.880336	1.261908

	A	B	C	D	E	F
0	10	-0.107567	0.069394	-1.845216	0.551087	-0.009382
1	9.3	-0.530735	0.471899	-0.415554	-0.557208	-1.476375
2	0.776062	-1.549687	0.880336	1.261908	-0.205181	1.775464

	A	B	C	D
0	10	-0.107567	0.069394	-1.845216
1	9.3	-0.530735	0.471899	-0.415554
2	0.776062	-1.549687	0.880336	1.261908

In [17]:

```
display(newdf1c.reset_index().head())
display(newdf1c.reset_index(drop=True).head())
```

	index	A	B	C	D
0	0	10	-0.107567	0.069394	-1.845216
1	1	9.3	-0.530735	0.471899	-0.415554
2	2	0.776062	-1.549687	0.880336	1.261908
3	3	-0.805187	0.619220	1.123326	0.273783
4	4	0.873255	0.200943	0.247609	0.978975

	A	B	C	D
0	10	-0.107567	0.069394	-1.845216
1	9.3	-0.530735	0.471899	-0.415554
2	0.776062	-1.549687	0.880336	1.261908
3	-0.805187	0.619220	1.123326	0.273783
4	0.873255	0.200943	0.247609	0.978975

In [18]:
```
display(newdf1c['B'].isnull())
newdf1c['D'] = None
display(newdf1c.head())
display(newdf1c.loc[:,['D']].isnull()) # or, newdf1c['D'].isnull()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
105    False
106    False
107    False
108    False
109    False
Name: B, Length: 110, dtype: bool
```

	A	B	C	D
0	10	-0.107567	0.069394	None
1	9.3	-0.530735	0.471899	None
2	0.776062	-1.549687	0.880336	None
3	-0.805187	0.619220	1.123326	None
4	0.873255	0.200943	0.247609	None

	D
0	True
1	True
2	True
3	True
4	True
...	...
105	True
106	True
107	True
108	True
109	True

110 rows × 1 columns

drop_duplicates :

In [19]:

```
display(newdf1c.dropna(how='all', axis=1).head())
newdf1c.loc[1:5, 'A'] = 1.43
display(newdf1c.head())
display(newdf1c.drop_duplicates(subset=['A'], keep='last'))
```

	A	B	C
0	10	-0.107567	0.069394
1	9.3	-0.530735	0.471899
2	0.776062	-1.549687	0.880336
3	-0.805187	0.619220	1.123326
4	0.873255	0.200943	0.247609

	A	B	C	D
0	10	-0.107567	0.069394	None
1	1.43	-0.530735	0.471899	None
2	1.43	-1.549687	0.880336	None
3	1.43	0.619220	1.123326	None
4	1.43	0.200943	0.247609	None

	A	B	C	D
0	10	-0.107567	0.069394	None
5	1.43	1.575089	-0.047594	None
6	0.562907	0.411450	-0.516616	None
7	-0.102692	-1.131892	-1.979243	None
8	0.129879	-0.733053	0.269876	None
...	...	...	...	...
105	-0.198477	1.091105	-1.489506	None
106	0.824008	-1.268946	-0.265850	None
107	-1.605694	-0.789346	-0.313562	None
108	0.861966	0.062173	-1.128290	None
109	0.796564	0.322604	-0.177676	None

106 rows × 4 columns

In [ ]:

Task:

Create a dataframe which contains only integers with 3 rows and 2 columns. Run the following methods on that dataframe:

1. df.count()
2. df.min()
3. df.max()
4. df.corr()
5. df.mean()
6. df.median()
7. df.std()
8. df.describe()

In [20]:

```
df2 = pd.DataFrame([[2,9],[8,3],[1,4]])
df2.columns = list('AB')
display(df2)
display('count', df2.count())
display('min', df2.min())
display('max', df2.max())
display('corr', df2.corr())
display('mean', df2.mean())
display('median', df2.median())
display('std', df2.std())
display('describe', df2.describe())
```

	A	B
0	2	9
1	8	3
2	1	4

```
'count'
```

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

```
'min'
```

```
A    1
B    3
dtype: int64
```

```
'max'
```

```
A    8
B    9
dtype: int64
```

```
'corr'
```

	A	B
A	1.000000	-0.520401
B	-0.520401	1.000000

```
'mean'
```

```
A    3.666667
B    5.333333
dtype: float64
```

```
'median'
```

```
A    2.0
B    4.0
dtype: float64
```

```
'std'
```

```
A    3.785939
B    3.214550
dtype: float64
```

```
'describe'
```

	A	B
count	3.000000	3.000000
mean	3.666667	5.333333
std	3.785939	3.214550
min	1.000000	3.000000
25%	1.500000	3.500000
50%	2.000000	4.000000
75%	5.000000	6.500000
max	8.000000	9.000000

Working with `Excel` files

In [21]:
```
cc09prog1 = pd.read_excel('cc09 modern physics prac prog1.xlsx', sheet_name='Sheet2')
display(cc09prog1.head())
display(cc09prog1.iloc[0,1])
cc09prog1.drop(['Unnamed: 0'], axis=1, inplace=True)
cc09prog1.drop([1,2,3], axis=0, inplace=True)
cc09prog1.columns = list([0,1,2,3,4,5,6,7,8])
display(cc09prog1.head(10))
```

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnan
0	NaN	Determination of Planck's constant using LEDs	NaN	NaN	NaN	NaN	NaN	NaN	I
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
2	NaN	2023-04-05 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	I
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
4	NaN	Color	Wavelength	Frequency	V_LED on	V_LED off	Mean V_threshold	Value of h	h ir

```
"Determination of Planck's constant using LEDs"
```

	0	1	2	3	4	5	6	7	8
0	Determination of Planck's constant using LEDs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Color	Wavelength	Frequency	V_LED on	V_LED off	Mean V_threshold	Value of h	h in J-s	Mean h
5	NaN	NaN	in e17 Hz	in V	in V	in V	in e-36 J-s	NaN	in J-s
6	Blue	480	0.00625	2.42	2.43	2.425	620.8	618.666667	580.295556
7	NaN	480	0.00625	2.41	2.4	2.405	615.68	NaN	NaN
8	NaN	480	0.00625	2.43	2.41	2.42	619.52	NaN	NaN
9	Green	560	0.005357	1.99	2.01	2	597.333333	602.808889	NaN
10	NaN	560	0.005357	2.01	2.03	2.02	603.306667	NaN	NaN
11	NaN	560	0.005357	2.01	2.06	2.035	607.786667	NaN	NaN
12	Yellow	590	0.005085	1.72	1.73	1.725	542.8	545.422222	NaN

In [22]:

```
lamb = cc09prog1.iloc[3:, 1].to_numpy()*1e-9
c, e = 3e8, 1.6e-19
freq = c/lamb
display('wavelength',lamb,'frequency',freq)
Vledon = cc09prog1.iloc[3:, 3].to_numpy()
Vledoff = cc09prog1.iloc[3:, 4].to_numpy()
Vthr = (Vledon + Vledoff)/2
display('V_led_on',Vledon,'V_led_off',Vledoff,'V_threshold',Vthr)
hexpt = e*Vthr/freq
hmean = hexpt.mean()
display('h',hexpt, 'mean h', hmean) # verified by pandas
```

```
'wavelength'

array([4.800000000000001e-07, 4.800000000000001e-07,
       4.800000000000001e-07, 5.6e-07, 5.6e-07, 5.6e-07,
       5.900000000000001e-07, 5.900000000000001e-07,
       5.900000000000001e-07, 6.350000000000001e-07,
       6.350000000000001e-07, 6.350000000000001e-07], dtype=object)

'frequency'

array([624999999999999.9, 624999999999999.9, 624999999999999.9,
       535714285714285.7, 535714285714285.7, 535714285714285.7,
       508474576271186.4, 508474576271186.4, 508474576271186.4,
       472440944881889.7, 472440944881889.7, 472440944881889.7],
      dtype=object)

'V_led_on'

array([2.42, 2.41, 2.43, 1.99, 2.01, 2.01, 1.72, 1.71, 1.75, 1.65, 1.62,
       1.64], dtype=object)

'V_led_off'

array([2.43, 2.4, 2.41, 2.01, 2.03, 2.06, 1.73, 1.75, 1.74, 1.64, 1.63,
       1.64], dtype=object)

'V_threshold'

array([2.425, 2.4050000000000002, 2.42, 2.0, 2.0199999999999996, 2.035,
       1.725, 1.73, 1.745, 1.645, 1.625, 1.64], dtype=object)

'h'

array([6.208e-34, 6.156800000000001e-34, 6.1952e-34,
       5.973333333333333e-34, 6.033066666666665e-34,
       6.077866666666667e-34, 5.428000000000001e-34,
       5.443733333333334e-34, 5.490933333333335e-34,
       5.571066666666674e-34, 5.503333333333333e-34,
       5.554133333333335e-34], dtype=object)

'mean h'

5.802955555555556e-34
```

In [ ]: