

▼ L3 Statistics with Pandas

▼ 1. What is Pandas?

Pandas is used as data cleaning module in the field of data science. You can do many operations in the dataset with this module. Can we clean or change the value in the dataset manually? We can if size of the dataset is small. What if we have a large dataset then we cannot do it manually it will take a lot of time. In this case, Pandas makes data manipulation very easy and effective.

▼ 2. Import the pandas module in your program

```
import pandas as pd
```

```
#Add code
```

```
!pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-package
```

```
import pandas as pd
```

▼ 3. Pandas Dataframe

▼ Read file into dataframe

```
d=pd.read_csv("path"):
```

- `pd.read_csv()` is the function to read the CSV(Comma separated values) file from your computer.
- In the function you have to pass a "path" and CSV filename.

```
d=pd.read_excel("path") :
```

- It is same as the `read_csv()` but it reads excel sheet or file.

Setup the Colab Drive with the path to the dataset

```
from google.colab import drive
drive.mount('/content/drive/')
datadir="/content/drive/My Drive/Data/DS3/"
```

#Add code

```
from google.colab import drive
drive.mount('/content/drive/')
datadir="/content/drive/My Drive/NYPITI02/L3/data/"
```

Drive already mounted at /content/drive/; to attempt to forcibly remount, call

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

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-4-00d18b01e5dd> in <module>()
      1 from google.colab import drive
----> 2 drive.mount(datadir)

----- 1 frames -----
/usr/local/lib/python3.7/dist-packages/google/colab/drive.py in _mount(mountpoint)
    120
    121     if ' ' in mountpoint:
--> 122         raise ValueError('Mountpoint must not contain a space.')
    123
    124     metadata_server_addr = _os.environ[

ValueError: Mountpoint must not contain a space.
```

SEARCH STACK OVERFLOW

Load the weather.csv using pandas read_csv() function

```
d=pd.read_csv(datadir+'weather.csv')
d
```

#Add code

```
d=pd.read_csv(datadir+'weather.csv')
# returning dataframe object
d
#printing dataframe d
```

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83
...
96448	2016-09-09 19:00:00.000 +0200	Partly Cloudy	rain	26.016667	26.016667	0.43
96449	2016-09-09 20:00:00.000 +0200	Partly Cloudy	rain	24.583333	24.583333	0.48
96450	2016-09-09 21:00:00.000 +0200	Partly Cloudy	rain	22.038889	22.038889	0.56

▼ Check how many rows in dataframe?

```
print(len(d))
# there are 96453 rows
```

#Add code

```
print(len(d))
# there are 96453 rows
```

96453

► Getting first n-rows of dataframe

- To see the first five rows call head() function with dataframe object.
for example-

```
d.head()
```

- If you want to view first n-rows

```
d.head(n)
```

```
[ ] ↪ 4 cells hidden
```

► Getting last n-rows of dataframe

- Call the tail() function

```
d.tail()
```

It will show only last five rows of dataframe d.

- If you want to see last n-rows -

```
d.tail(n)
```

```
[ ] ↪ 4 cells hidden
```

► Slicing the dataframe

- Suppose you want to get 10 rows of the dataframe ranging from row 20 to 30.

```
d[20:30+1]
```

```
[ ] ↪ 2 cells hidden
```

▼ Accessing the particular column of dataframe :

- You can access the particular column of the dataframe
for example you want to access Humidity column

```
d['Humidity']
```

#Add code

```
d['Humidity'].head(10)
```

```
#Here i have applied head() function just to see only first 10 values because humid  
#column has too many values
```

```
0    0.89
1    0.86
2    0.89
3    0.83
4    0.83
5    0.85
6    0.95
7    0.89
8    0.82
9    0.72
```

```
Name: Humidity, dtype: float64
```

► Finding min, max and average of a column:

- **min()** : To find minimum of a column
- **max()** : To find maximum of a column
- **mean()** : To find average of a column

Find minimum and maximum of Humidity column-

[] ↪ 9 cells hidden

► Conditional statements

- `d["your column"]["your condition"]`
It will return all the values in which your condition holds true.

Examples :

- Find temp when Humidity is minimum

```
d['Temperature(C)'][d['Humidity']==d['Humidity'].min() ]
```

- Find temp when Humidity is maximum

```
d['Temperature(C)'][d['Humidity']==d['Humidity'].max() ]
```

[] ↪ 4 cells hidden

► 4. Statistical Information using dataframe

[] ↪ 9 cells hidden

► 5. Handling Missing Data

- `fillna()` method
- Forward fill method
- Backward fill method
- `dropna()` method

[] ↪ 31 cells hidden

► 6. Groupby for dataframe

Groupby is one of the important operations in data analysis

Groupby property is grouped the data according the column supplied to the function.

[] ↪ 19 cells hidden

▼ 7. Visualisation of Data

We can display some of the common graph using the dataframe plot functions

```
import pandas as pd
```

```
df = pd.DataFrame({
    'name': ['john', 'mary', 'peter', 'jeff', 'bill', 'lisa', 'jose'],
    'age': [23, 78, 22, 19, 45, 33, 20],
    'gender': ['M', 'F', 'M', 'M', 'M', 'F', 'M'],
    'state': ['california', 'dc', 'california', 'dc', 'california', 'texas', 'texas'],
    'num_children': [2, 0, 0, 3, 2, 1, 4],
    'num_pets': [5, 1, 0, 5, 2, 2, 3]
})
df
```

	name	age	gender	state	num_children	num_pets
0	john	23	M	california	2	5
1	mary	78	F	dc	0	1
2	peter	22	M	california	0	0
3	jeff	19	M	dc	3	5
4	bill	45	M	california	2	2
5	lisa	33	F	texas	1	2
6	jose	20	M	texas	4	3

► Scatter Plot

Display a scatter plot compare 2 columns

Dispaly scatter plot comparing num_children and num_pets

```
df.plot(kind='scatter', x='num_children', y='num_pets', color='red')
plt.show()
```

[] ↪ 2 cells hidden

► Bar Chart





Display a bar chart to show the number of pets each person is owned.

```
df.plot(kind='bar', x='name', y='num_pets')
```

[] ↪ 2 cells hidden

▼ 8. map(), apply(), applymap()

Fast operation on column of data

	DataFrame	Series
apply		
map		
applymap		

```
df = pd.read_csv(datadir+"p3_data2.tsv", "\t")
df
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882:
exec(code_obj, self.user_global_ns, self.user_ns)
```

	ID	Advertising costs	Part-time Labour costs
0	1	362438	8763
1	2	51725	9258
2	3	236472	9897
3	4	232080	5005
4	5	262733	9918
5	6	355115	9996
6	7	237687	7732
7	8	57348	7378
8	9	191714	7243
9	10	190431	5388
10	11	62646	6965
11	12	146397	5887
12	13	257691	6016
13	14	77524	7558

▼ map() function

Find the data in the column 'Advertising costs' > 200000

Store the result as either True or False into a new column "Is High Cost"

Insert the "Is High Cost" column into the dataframe

```
df_clone = df.copy()
df_clone["Is Low Cost"] = df['Advertising costs'].map(lambda x: x>200000)
df_clone.head()
```

22	23	300124	7279
----	----	--------	------

To apply a common operation to all values in a Series, we can use the `.map()` which is similar to `map()` for python lists.

#Add code

20	21	104430	3320
----	----	--------	------

#Add code

```
df["Is Low Cost"] = df['Advertising costs'].map(lambda x: x>200000)
df.head()
```


	ID	Advertising costs	Part-time Labour costs	Is Low Cost
0	1	362438	8763	True
1	2	51725	9258	False
2	3	236472	9897	True
3	4	232080	5005	True
4	5	262733	9918	True

▼ apply function

`.apply(f)` is a method which performs column-wise aggregation on a dataframe.

Get the two column 'Advertising costs','Part-time Labour costs' then apply a normalization by divide the data in the 2 column by 1000.

```
def normalize(x):
    a= x /1000
    return a
df1=df[['Advertising costs','Part-time Labour costs']].apply(normalize)
df1
```

#Add code

```
def normalize(x):
    a= x /1000
    return a
df1=df[['Advertising costs','Part-time Labour costs']].apply(normalize)
df1
```

	Advertising costs	Part-time Labour costs
0	362.438	8.763
1	51.725	9.258
2	236.472	9.897
3	232.080	5.005
4	262.733	9.918
5	355.115	9.996
6	237.687	7.732
7	57.348	7.378
8	191.714	7.243
9	190.431	5.388
10	62.646	6.965
11	146.397	5.887
12	257.691	6.016
13	77.524	7.558
14	294.566	5.517
15	149.813	6.860
16	242.242	8.382
17	122.575	9.059
18	63.018	6.688

▼ applymap function

Convert all the data in column 'Advertising costs','Part-time Labour costs' into negative value.

```
df1.applymap(lambda x:-x).head()
```

```
04      362.438      8.763
#Add code
```

```
df1.applymap(lambda x:-x).head()
```

	Advertising costs	Part-time Labour costs
0	-362.438	-8.763
1	-51.725	-9.258
2	-236.472	-9.897
3	-232.080	-5.005
4	-262.733	-9.918
39	363.884	6.408
40	129.481	8.913
41	247.943	9.876
42	273.692	5.115
43	27.551	6.820

