Pandas Tutorial

• Pandas are mostly used in data science to manipulate and analyze the data

• Easy Manipulation and simple representation of large datasets

• Easy filtering, segmentation and segregation of Datasets

Chapter 1

In [1]:
```
#import the panda library
import pandas as pd

#reading tsv format datasets (tab separated values)
emp = pd.read_table('C:/Users/Rasha/Downloads/tsv_sample.tsv')
emp.head()
```

Out[1]:

	Name	Position	Office	Age	Start date	Salary
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
1	Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	$1,200,000
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000
3	Bradley Greer	Software Engineer	London	41	2012/10/13	$132,000
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850

In [2]:
```
# For pipe separated delimiters(syntax : name|age|group) follow the below

pipe_sv = pd.read_table('https://www.nrc.gov/reading-rm/doc-collections/ev
pipe_sv.head() #head means it shows only the first few(first 5) instances

# op: displays a dataset of the table format
```

Out[2]:

	ReportDt	Unit	Power
0	12/31/2009	Arkansas Nuclear 1	100
1	12/31/2009	Arkansas Nuclear 2	100
2	12/31/2009	Beaver Valley 1	100

	ReportDt	Unit	Power
3	12/31/2009	Beaver Valley 2	100
4	12/31/2009	Braidwood 1	100

In [3]:
```
pipe_sv #displays the whole dataset
```

Out[3]:

	ReportDt	Unit	Power
0	12/31/2009	Arkansas Nuclear 1	100
1	12/31/2009	Arkansas Nuclear 2	100
2	12/31/2009	Beaver Valley 1	100
3	12/31/2009	Beaver Valley 2	100
4	12/31/2009	Braidwood 1	100
...	...	...	...
37955	01/01/2009	Vogtle 1	100
37956	01/01/2009	Vogtle 2	100
37957	01/01/2009	Waterford 3	100
37958	01/01/2009	Watts Bar 1	100
37959	01/01/2009	Wolf Creek 1	100

37960 rows × 3 columns

Note : Overall I've taken two different datasets here. One for tab separated values and other for pipe delimetered values

Chapter 2

In [4]:
```
# selecting certain colums
# here i am taking the first data set


cols = ['Name', 'Position']
emp = pd.read_table('C:/Users/Rasha/Downloads/tsv_sample.tsv')
emp[cols].head()
```

Out[4]:

	Name	Position
0	Airi Satou	Accountant
1	Angelica Ramos	Chief Executive Officer (CEO)
2	Ashton Cox	Junior Technical Author

	Name	Position
3	Bradley Greer	Software Engineer
4	Brenden Wagner	Software Engineer

In [5]:

```
# Selecting certain rows

emp = pd.read_table('C:/Users/Rasha/Downloads/tsv_sample.tsv', nrows = 10)
emp
```

Out[5]:

	Name	Position	Office	Age	Start date	Salary
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
1	Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	$1,200,000
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000
3	Bradley Greer	Software Engineer	London	41	2012/10/13	$132,000
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850
5	Brielle Williamson	Integration Specialist	New York	61	2012/12/02	$372,000
6	Bruno Nash	Software Engineer	London	38	2011/05/03	$163,500
7	Caesar Vance	Pre-Sales Support	New York	21	2011/12/12	$106,450
8	Cara Stevens	Sales Assistant	New York	46	2011/12/06	$145,600
9	Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	$433,060

In [6]:

```
#To find the datatypes of the columns

emp = pd.read_table('C:/Users/Rasha/Downloads/tsv_sample.tsv') #displays t
emp.dtypes
```

Out[6]:

```
Name          object
Position      object
Office        object
Age            int64
Start date    object
Salary        object
dtype: object
```

In [7]:

```
# Display integer datatypes

import numpy as np
emp = pd.read_table('C:/Users/Rasha/Downloads/tsv_sample.tsv')
emp.select_dtypes(include = [np.number]).dtypes #numpy is used cause mathe
```

Out[7]:
```
Age     int64
dtype: object
```

In built data analysis in pandas

In [8]:
```
# Now our emp just has the age column (with all int instances)

emp.describe() # briefly describes the data. That is number of cols, mean

#the describe operation can only be performed on numeric values
```

Out[8]:

	Age
count	57.000000
mean	42.736842
std	14.877507
min	19.000000
25%	30.000000
50%	42.000000
75%	56.000000
max	66.000000

In [9]:
```
# tells the number of rows and columns

emp.shape
```

Out[9]:
```
(57, 6)
```

In [10]:
```
type(emp) #DataFrame is of the table format (2D form), like a csv excel
```

Out[10]:
```
pandas.core.frame.DataFrame
```

Chapter 3

In [11]:
```
#Concatenating columns

emp = pd.read_table('C:/Users/Rasha/Downloads/tsv_sample.tsv')
emp['Name Salary'] = emp['Name']+emp['Salary'] #concatenates name and sala
emp.head()
```

Out[11]:

	Name	Position	Office	Age	Start date	Salary	Name Salary
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700	Airi Satou$162,700

	Name	Position	Office	Age	Start date	Salary	Name Salary
1	Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	$1,200,000	Angelica Ramos$1,200,000
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000	Ashton Cox$86,000
3	Bradley Greer	Software Engineer	London	41	2012/10/13	$132,000	Bradley Greer$132,000
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850	Brenden Wagner$206,850

In [12]:

```
#dropping columns

emp = emp.drop('Name Salary',axis=1) #axis=1 represents its a column
emp.head()
```

Out[12]:

	Name	Position	Office	Age	Start date	Salary
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
1	Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	$1,200,000
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000
3	Bradley Greer	Software Engineer	London	41	2012/10/13	$132,000
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850

In [13]:

```
# Renaming certain columns

ren_col = ['Name','Position','Ofc','Age','StartDate','Sal']
emp.columns = ren_col #keyword columns is used. (Hint: Press tab after a k
emp.head()
```

Out[13]:

	Name	Position	Ofc	Age	StartDate	Sal
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
1	Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	$1,200,000
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000
3	Bradley Greer	Software Engineer	London	41	2012/10/13	$132,000
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850

• Sorting Data and Filtering

In [14]:
```
#Sorting by data(sorting the table by age)

sort_asc = emp.sort_values(by='Age',ascending=True) #Sorts the data in asc
sort_asc.head()
```

Out[14]:

	Name	Position	Ofc	Age	StartDate	Sal
48	Tatyana Fitzpatrick	Regional Director	London	19	2010/03/17	$385,750
45	Shou Itou	Regional Marketing	Tokyo	20	2011/08/14	$163,000
7	Caesar Vance	Pre-Sales Support	New York	21	2011/12/12	$106,450
32	Lael Greer	Systems Administrator	London	21	2009/02/27	$103,500
18	Gavin Cortez	Team Leader	San Francisco	22	2008/10/26	$235,500

In [15]:
```
#Sorting a series

sort_series = emp['Name'].sort_values() #just displays a column in its sor
sort_series.head()
```

Out[15]:
```
0        Airi Satou
1    Angelica Ramos
2        Ashton Cox
3     Bradley Greer
4    Brenden Wagner
Name: Name, dtype: object
```

In [16]:
```
# Filtering (Just like the where clause)

sort_byage = emp[emp.Age < 40] #displays all those data aged below 40
sort_byage.head()
```

Out[16]:

	Name	Position	Ofc	Age	StartDate	Sal
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850
6	Bruno Nash	Software Engineer	London	38	2011/05/03	$163,500
7	Caesar Vance	Pre-Sales Support	New York	21	2011/12/12	$106,450
9	Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	$433,060

In [17]:
```
# Multifiltering Criteria

mul_fil = emp[(emp.Age < 40) & (emp.Name == "Airi Satou")]
mul_fil
```

Out[17]:

	Name	Position	Ofc	Age	StartDate	Sal
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700

In [18]:

```
mul_col = ['Name','Age']
emp[emp.Age < 40][mul_col].head()
```

Out[18]:

	Name	Age
0	Airi Satou	33
4	Brenden Wagner	28
6	Bruno Nash	38
7	Caesar Vance	21
9	Cedric Kelly	22

Chapter 4

• Mean

In [20]:

```
emp.mean() #since there is only one integer column, hence only one mean is
```

Out[20]:

```
Age     42.736842
dtype: float64
```

• String Manipulation Techniques

In [25]:

```
#to convert string to lowercase

emp.Name.str.lower().head()

# to convert string into uppercase

emp.Name.str.upper().head()
```

Out[25]:

```
0         AIRI SATOU
1    ANGELICA RAMOS
2         ASHTON COX
3      BRADLEY GREER
4    BRENDEN WAGNER
Name: Name, dtype: object
```

In [27]:

```
# To check which of the rows of column position has the keyword 'Software'

emp.Position.str.contains('Software') #displays all the rows of the datase

emp[emp.Position.str.contains('Software')]
```

Out[27]:

	Name	Position	Ofc	Age	StartDate	Sal
3	Bradley Greer	Software Engineer	London	41	2012/10/13	$132,000
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850
6	Bruno Nash	Software Engineer	London	38	2011/05/03	$163,500
46	Sonya Frost	Software Engineer	Edinburgh	23	2008/12/13	$103,600
55	Zenaida Frank	Software Engineer	New York	63	2010/01/04	$125,250
56	Zorita Serrano	Software Engineer	San Francisco	56	2012/06/01	$115,000

In [29]:
```
# To replace a word by another in the dataset

emp.Position.str.replace('Engineer','Developer').head()
```

Out[29]:
```
0                       Accountant
1    Chief Executive Officer (CEO)
2          Junior Technical Author
3                Software Developer
4                Software Developer
Name: Position, dtype: object
```

• Aggregation and Group By Clause

In [31]:
```
emp.Age.min() #the min age in the dataset
```

Out[31]:
```
19
```

In [32]:
```
emp.Age.max() #the max value of Age in the dataset
```

Out[32]:
```
66
```

In [34]:
```
#Using group by clause
emp.groupby('Position').Age.min() #Since here group by clause is used, and
```

Out[34]:
```
Position
Accountant                        33
Chief Executive Officer (CEO)     47
Chief Financial Officer (CFO)     64
Chief Marketing Officer (CMO)     40
Chief Operating Officer (COO)     48
Customer Support                  27
Data Coordinator                  64
Developer                         30
Development Lead                  30
Director                          65
Financial Controller              62
Integration Specialist            37
Javascript Developer              29
Junior Javascript Developer       43
Junior Technical Author           66
```

```
Marketing Designer                47
Office Manager                    30
Personnel Lead                    35
Post-Sales support                46
Pre-Sales Support                 21
Regional Director                 19
Regional Marketing                20
Sales Assistant                   23
Secretary                         41
Senior Javascript Developer       22
Senior Marketing Designer         43
Software Engineer                 23
Support Engineer                  37
Support Lead                      22
System Architect                  61
Systems Administrator             21
Team Leader                       22
Technical Author                  27
Name: Age, dtype: int64
```

In [35]:

```
# For a better representation we use aggregate

emp.groupby('Position').Age.agg(['count','min','max'])
```

Out[35]:

Position	count	min	max
Accountant	2	33	63
Chief Executive Officer (CEO)	1	47	47
Chief Financial Officer (CFO)	1	64	64
Chief Marketing Officer (CMO)	1	40	40
Chief Operating Officer (COO)	1	48	48
Customer Support	1	27	27
Data Coordinator	1	64	64
Developer	4	30	61
Development Lead	1	30	30
Director	1	65	65
Financial Controller	1	62	62
Integration Specialist	3	37	61
Javascript Developer	2	29	39
Junior Javascript Developer	1	43	43
Junior Technical Author	1	66	66
Marketing Designer	2	47	66
Office Manager	3	30	51
Personnel Lead	1	35	35

Position	count	min	max
Post-Sales support	1	46	46
Pre-Sales Support	1	21	21
Regional Director	5	19	51
Regional Marketing	1	20	20
Sales Assistant	3	23	59
Secretary	1	41	41
Senior Javascript Developer	1	22	22
Senior Marketing Designer	1	43	43
Software Engineer	6	23	63
Support Engineer	3	37	64
Support Lead	1	22	22
System Architect	1	61	61
Systems Administrator	2	21	59
Team Leader	1	22	22
Technical Author	1	27	27

Chapter 5

• Using loc

In [38]:
```
emp = pd.read_table('C:/Users/Rasha/Downloads/tsv_sample.tsv')
emp.head()
emp.loc[0:2,:] #Displays rows 0 to 2 and all columns
```

Out[38]:

	Name	Position	Office	Age	Start date	Salary
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
1	Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	$1,200,000
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000

In [42]:
```
# To display rows 0-5 and first 3 columns

emp.loc[0:5,'Name':'Office'] #cannot do slice indexing on Index with type
```

Out[42]:

	Name	Position	Office
0	Airi Satou	Accountant	Tokyo
1	Angelica Ramos	Chief Executive Officer (CEO)	London
2	Ashton Cox	Junior Technical Author	San Francisco
3	Bradley Greer	Software Engineer	London
4	Brenden Wagner	Software Engineer	San Francisco
5	Brielle Williamson	Integration Specialist	New York

In [45]:
```
# Rows with certain conditions

emp.loc[emp.Position=='Software Engineer','Name':'Position'] #Dsiplays onl
```

Out[45]:

	Name	Position
3	Bradley Greer	Software Engineer
4	Brenden Wagner	Software Engineer
6	Bruno Nash	Software Engineer
46	Sonya Frost	Software Engineer
55	Zenaida Frank	Software Engineer
56	Zorita Serrano	Software Engineer

• Using dropna (Droppping rows with missing values)

In [47]:
```
emp_miss = pd.read_table('C:/Users/Rasha/Downloads/miss_tsv.tsv')
emp_miss.head()
```

Out[47]:

	Name	Position	Office	Age	Start date	Salary
0	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
1	Angelica Ramos	Chief Executive Officer (CEO)	London	47	2009/10/09	NaN
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	NaN
3	Bradley Greer	Software Engineer	London	41	2012/10/13	$132,000
4	Brenden Wagner	Software Engineer	San Francisco	28	2011/06/07	$206,850

In [51]:
```
emp.shape  #Shape of the original dataset (57 rows and 6 columns)
```

Out[51]:
```
(57, 6)
```

In [52]:
```
emp_miss.dropna(how='any').shape #shape after dropping the rows having atl
```

Out[52]:
```
(55, 6)
```

In [55]:
```
# Dropping the rows with certain conditions

emp_miss.dropna(subset=['Name','Salary'],how='any').shape #any row having
```

Out[55]:
```
(55, 6)
```

Chapter 6

• Working with plots

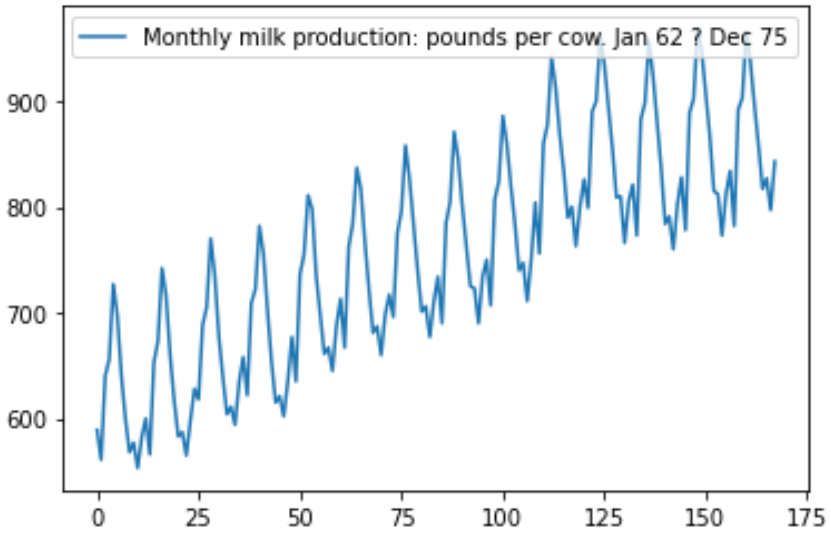
In [57]:
```
#Plotting graphs purely through pandas library

data = pd.read_csv('C:/Users/Rasha/Downloads/mon_milk.csv')
data.head()
```

Out[57]:

	Month	Monthly milk production: pounds per cow. Jan 62 ? Dec 75
0	1962-01	589.0
1	1962-02	561.0
2	1962-03	640.0
3	1962-04	656.0
4	1962-05	727.0

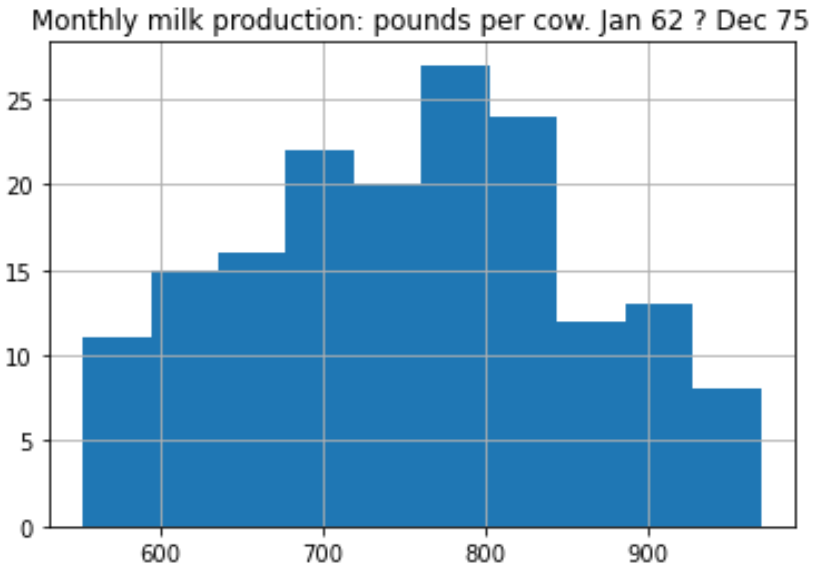
In [58]:
```
#Since there are only 2 columns pandas predicts that the 1st column must b

data.plot() # Randomly plotting data (by default : line graph)
```

Out[58]:
```
<AxesSubplot:>
```



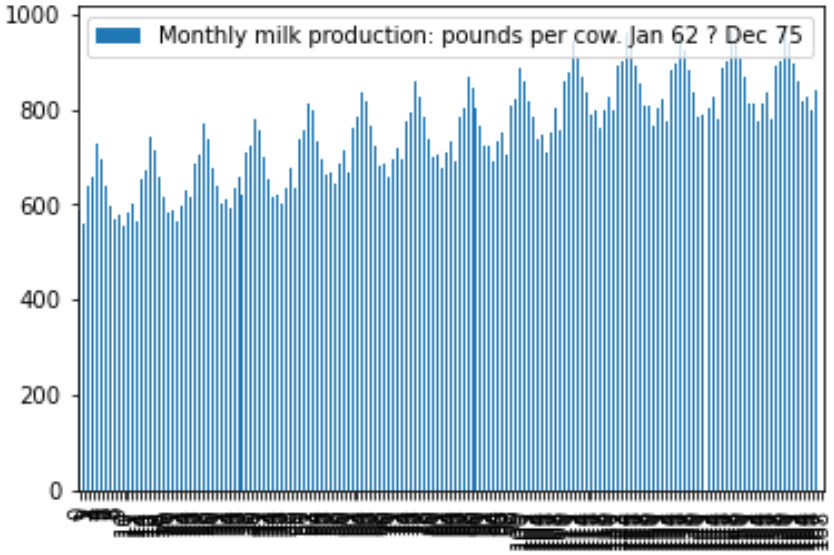
In [59]:
```
data.hist() #to plot a histogram for the given dataset
```

Out[59]:
```
array([[<AxesSubplot:title={'center':'Monthly milk production: pounds per
cow. Jan 62 ? Dec 75'}>]],
      dtype=object)
```



In [66]:
```
data.plot(kind='bar') #defining the kind of graph you want using the keywo
```

Out[66]:
```
<AxesSubplot:>
```



• Joins

In [68]:
```
# Creating a table

data_sm1 = pd.DataFrame({  #Data frame is a datatype that defines the data
    "Student" : ["Akash","Joe","Sony"],
    "Age" :["15","17","8"]
})

data_sm1
```

Out[68]:

	Student	Age
0	Akash	15
1	Joe	17
2	Sony	8

In [71]:
```
data_sm2 = pd.DataFrame({
    "Student" : ["Akash","Joe","Muna"],
    "Marks" : ["80","45","98"]
})

data_sm2
```

Out[71]:

	Student	Marks
0	Akash	80
1	Joe	45
2	Muna	98

In [72]:
```
# Inner Join

pd.merge(data_sm1,data_sm2,on="Student") #inner join by default
```

```
# Inner join finds the similarity in both the datasets . Here we have set
```

Out[72]:

	Student	Age	Marks
0	Akash	15	80
1	Joe	17	45

In [74]:

```
#Outer Join

pd.merge(data_sm1,data_sm2,on="Student", how="outer")

#Outer join merges all the data in both the datasets and displays them
```

Out[74]:

	Student	Age	Marks
0	Akash	15	80
1	Joe	17	45
2	Sony	8	NaN
3	Muna	NaN	98

In [79]:

```
# Left Join

# Considers all the data on the left sided dataset (here left sided datset
pd.merge(data_sm1,data_sm2,on="Student", how="left")
```

Out[79]:

	Student	Age	Marks
0	Akash	15	80
1	Joe	17	45
2	Sony	8	NaN

In [80]:

```
# Right Join

# Considers all the data on the right sided dataset (here right sided dats
pd.merge(data_sm1,data_sm2,on="Student", how="right")
```

Out[80]:

	Student	Age	Marks
0	Akash	15	80
1	Joe	17	45
2	Muna	NaN	98

• Pivoting

In [94]:
```
piv_sam = pd.read_csv('C:/Users/Rasha/Downloads/pivot_sample.csv')
piv_sam
```

Out[94]:

	Location	Page	Hits
0	Mumbai	ABC	2
1	Mumbai	PQR	12
2	Mumbai	MNO	54
3	Mumbai	XYZ	86
4	Goa	ABC	75
5	Goa	PQR	100
6	Goa	MNO	25
7	Goa	XYZ	65
8	Hawai	ABC	98
9	Hawai	PQR	444
10	Hawai	MNO	10
11	Hawai	XYZ	55

In [96]:
```
# Pivot the data

piv_sam.pivot(index ="Page",columns="Location") #Selecting index means sel

#The o/p will show the hits of a particular page at different location (co
```

Out[96]:

	Hits		
Location	**Goa**	**Hawai**	**Mumbai**
Page			
ABC	75	98	2
MNO	25	10	54
PQR	100	444	12
XYZ	65	55	86

In [100…
```
#Pivot the table

piv_sam.pivot_table(index = "Page", aggfunc = "sum")

#finds and calculates the total number of hits for every page (ignored att
```

Out[100…

	Hits
Page	

Page	Hits
ABC	175
MNO	89
PQR	556
XYZ	206

In [147...

```
piv_mean_loc = piv_sam.pivot_table(index = "Location", aggfunc = "mean")

#finds the mean of total number of hits for every location (ignored attrib
piv_mean_loc

#for later reference ive created this variable piv_mean_loc
```

Out[147...

Location	Hits
Goa	66.25
Hawai	151.75
Mumbai	38.50

In [143...

```
piv_sam.pivot_table(index = "Location", aggfunc="count") #no ignored attri
```

Out[143...

Location	Hits	Page
Goa	4	4
Hawai	4	4
Mumbai	4	4

In [144...

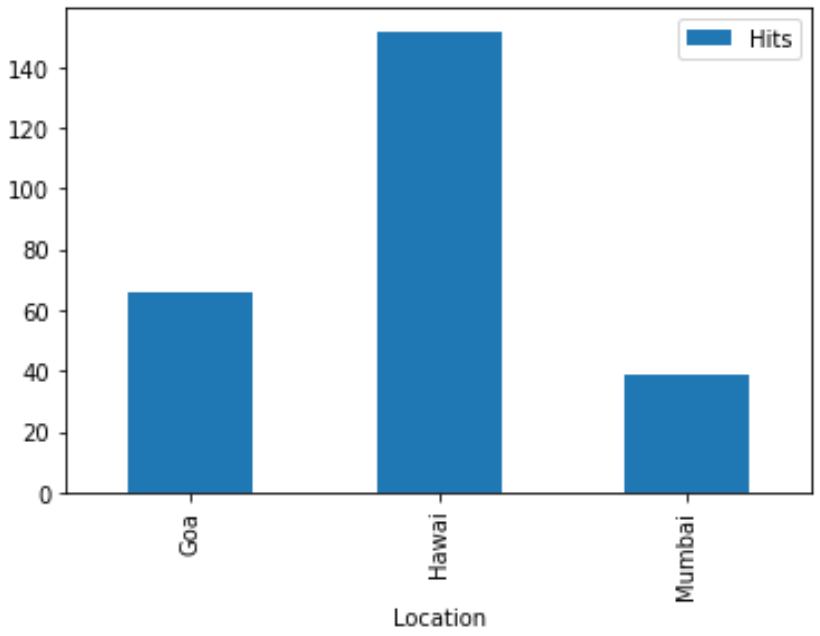
```
#To find the grand total of the column 'hits'

piv_sam['Hits'].sum()
```

Out[144... 1026

In [146...

```
# Plotting a bar chart that describes the number of hits at different loca

piv_mean_loc.plot(kind = 'bar')

#note : the piv_mean_loc describes the mean of total hits at locations GOA
```

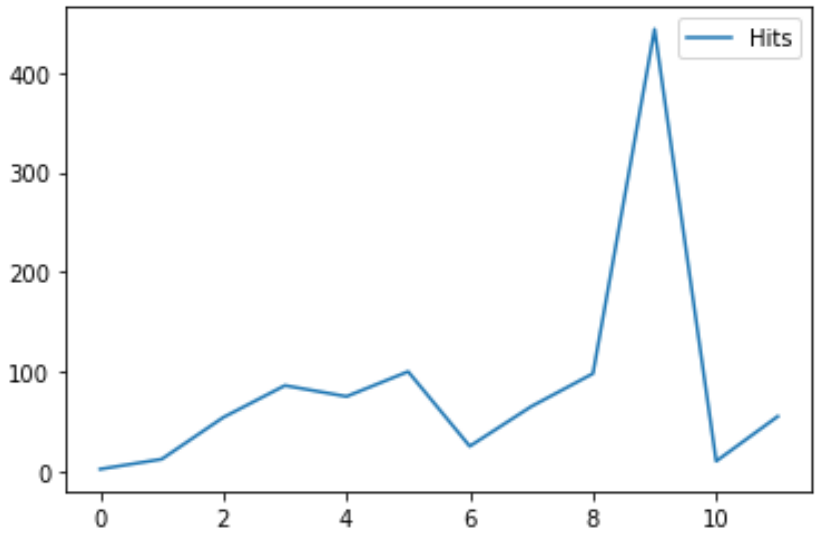
Out[146… <AxesSubplot:xlabel='Location'>



In [152…

```
#Line chart of total hits vs the index(0-11)

piv_sam.plot(kind = 'line')
```

Out[152… <AxesSubplot:>



Chapter 6

• Shifting

In [157…

```
# Shifting the row by one position down

fb_stock = pd.read_csv('C:/Users/Rasha/Downloads/FB.csv')
fb_stock.head()

fb_stock.shift(1).head()
```

Out[157...

	Date	Open	High	Low	Close	Adj Close	Volume
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2018-06-27	199.179993	200.750000	195.800003	195.839996	195.839996	18734400.0
2	2018-06-28	195.179993	197.339996	193.259995	196.229996	196.229996	18172400.0
3	2018-06-29	197.320007	197.600006	193.960007	194.320007	194.320007	15811600.0
4	2018-07-02	193.369995	197.449997	192.220001	197.360001	197.360001	13961600.0

In [158...

```
# Shifting the row by one position above

fb_stock.shift(-1).head()
```

Out[158...

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-06-28	195.179993	197.339996	193.259995	196.229996	196.229996	18172400.0
1	2018-06-29	197.320007	197.600006	193.960007	194.320007	194.320007	15811600.0
2	2018-07-02	193.369995	197.449997	192.220001	197.360001	197.360001	13961600.0
3	2018-07-03	194.550003	195.399994	192.520004	192.729996	192.729996	13489500.0
4	2018-07-05	194.740005	198.649994	194.029999	198.449997	198.449997	19684200.0

In [160...

```
#Calculating the prvious closing price.(for the 2nd row take the value of

fb_stock['prev_close'] = fb_stock['Close'].shift(1)

fb_stock.head()
```

Out[160...

	Date	Open	High	Low	Close	Adj Close	Volume	prev_cl
0	2018-06-27	199.179993	200.750000	195.800003	195.839996	195.839996	18734400	N
1	2018-06-28	195.179993	197.339996	193.259995	196.229996	196.229996	18172400	195.839
2	2018-06-29	197.320007	197.600006	193.960007	194.320007	194.320007	15811600	196.229
3	2018-07-02	193.369995	197.449997	192.220001	197.360001	197.360001	13961600	194.320

	Date	Open	High	Low	Close	Adj Close	Volume	prev_cl
4	2018-07-03	194.550003	195.399994	192.520004	192.729996	192.729996	13489500	197.360

In [161…

```
# To build a column that depicts the diff bw close column values and the p

fb_stock['diff_close'] = fb_stock['Close']-fb_stock['prev_close']

fb_stock.head()
```

Out[161…

	Date	Open	High	Low	Close	Adj Close	Volume	prev_cl
0	2018-06-27	199.179993	200.750000	195.800003	195.839996	195.839996	18734400	N
1	2018-06-28	195.179993	197.339996	193.259995	196.229996	196.229996	18172400	195.839
2	2018-06-29	197.320007	197.600006	193.960007	194.320007	194.320007	15811600	196.229
3	2018-07-02	193.369995	197.449997	192.220001	197.360001	197.360001	13961600	194.320
4	2018-07-03	194.550003	195.399994	192.520004	192.729996	192.729996	13489500	197.360

In [166…

```
# Generating Weekly returns

fb_stock['weeky_returns'] = ((fb_stock['Close']-fb_stock['prev_close'].shi
# Generates the weekly returns hence the first seven rows are empty(shift
fb_stock.head()
```

Out[166…

	Date	Open	High	Low	Close	Adj Close	Volume	prev_cl
0	2018-06-27	199.179993	200.750000	195.800003	195.839996	195.839996	18734400	N
1	2018-06-28	195.179993	197.339996	193.259995	196.229996	196.229996	18172400	195.839
2	2018-06-29	197.320007	197.600006	193.960007	194.320007	194.320007	15811600	196.229
3	2018-07-02	193.369995	197.449997	192.220001	197.360001	197.360001	13961600	194.320
4	2018-07-03	194.550003	195.399994	192.520004	192.729996	192.729996	13489500	197.360

In [167…

```
# Writing to an excel sheeft/csv file. (i.e exporting the csv file from no
# Here our data is stored in varible fb_stock

fb_stock.to_csv('C://Users/Rasha/Downloads/fb_exported.csv' #you can also
```