Data Frame

In [8]:
```
# All imports
import numpy as np
import pandas as pd
```

Example - 1

Create a Data Frame cars using raw data stored in a dictionary

In [9]:
```
cars_per_cap = [809, 731, 588, 18, 200, 70, 45]
country = ['United States', 'Australia', 'Japan', 'India', 'Russia', 'Morocco', 'Egypt']
drives_right = [True, False, False, False, True, True, True]
```

In [10]:
```
data = {"cars_per_cap": cars_per_cap, "country": country, "drives_right": drives_right}
```

In [11]:
```
data
```

Out[11]:
```
{'cars_per_cap': [809, 731, 588, 18, 200, 70, 45],
 'country': ['United States',
  'Australia',
  'Japan',
  'India',
  'Russia',
  'Morocco',
  'Egypt'],
 'drives_right': [True, False, False, False, True, True, True]}
```

In [12]:
```
cars = pd.DataFrame(data)

cars
```

Out[12]:

	cars_per_cap	country	drives_right
0	809	United States	True
1	731	Australia	False
2	588	Japan	False
3	18	India	False
4	200	Russia	True
5	70	Morocco	True
6	45	Egypt	True

In [13]:
```
type(cars)
```

Out[13]:
```
pandas.core.frame.DataFrame
```

Example - 2 (Reading data from a file)

Create a Data Frame by importing cars data from cars.csv

In [16]:
```
# Read a file using pandas

cars_df = pd.read_csv('cars.csv')

cars_df
```

Out[16]:

	USCA	US	United States	809	FALSE
0	ASPAC	AUS	Australia	731.0	True
1	ASPAC	JAP	Japan	588.0	True
2	ASPAC	IN	India	18.0	True
3	ASPAC	RU	Russia	200.0	False
4	LATAM	MOR	Morocco	70.0	False
5	AFR	EG	Egypt	45.0	False
6	EUR	ENG	England	NaN	True

Example - 3 (Column headers)

Read file - skip header

In [35]:
```
cars_df = pd.read_csv('cars.csv', header=None)

cars_df
```

Out[35]:

	0	1	2	3	4
0	USCA	US	United States	809.0	False
1	ASPAC	AUS	Australia	731.0	True
2	ASPAC	JAP	Japan	588.0	True
3	ASPAC	IN	India	18.0	True
4	ASPAC	RU	Russia	200.0	False
5	LATAM	MOR	Morocco	70.0	False
6	AFR	EG	Egypt	45.0	False
7	EUR	ENG	England	NaN	True

Assign Headers

In [36]:
```
# Returns an array of headers

cars_df.columns
```

Out[36]:
```
Int64Index([0, 1, 2, 3, 4], dtype='int64')
```

In [37]:
```
# Rename Headers

cars_df.columns = ['country code', 'region', 'country', 'cars_per_cap', 'drive_right']
```

In [38]:
```
cars_df
```

Out[38]:

	country code	region	country	cars_per_cap	drive_right
0	USCA	US	United States	809.0	False
1	ASPAC	AUS	Australia	731.0	True
2	ASPAC	JAP	Japan	588.0	True
3	ASPAC	IN	India	18.0	True
4	ASPAC	RU	Russia	200.0	False
5	LATAM	MOR	Morocco	70.0	False
6	AFR	EG	Egypt	45.0	False
7	EUR	ENG	England	NaN	True

Example - 4 (Row index/names)

Read file - skip header and assign first column as index.

In [31]:
```
# Index is returned by
cars_df.index
```

Out[31]:
```
RangeIndex(start=0, stop=8, step=1)
```

In [43]:
```
# Read file and set 1st column as index
cars_df = pd.read_csv("cars.csv", header= None, index_col=0)

# set the column names
cars_df.columns = ['region', 'country', 'cars_per_cap', 'drive_right']
cars_df
```

Out[43]:

	region	country	cars_per_cap	drive_right
0				
USCA	US	United States	809.0	False
ASPAC	AUS	Australia	731.0	True
ASPAC	JAP	Japan	588.0	True
ASPAC	IN	India	18.0	True
ASPAC	RU	Russia	200.0	False
LATAM	MOR	Morocco	70.0	False
AFR	EG	Egypt	45.0	False
EUR	ENG	England	NaN	True

In [44]:
```
# Print the new index
cars_df.index
```

Out[44]:
```
Index(['USCA', 'ASPAC', 'ASPAC', 'ASPAC', 'ASPAC', 'LATAM', 'AFR', 'EUR'], dtype='object', name=0)
```

Rename the Index Name

In [46]:
```
cars_df.index.name = 'country_code'
cars_df
```

Out[46]:

	region	country	cars_per_cap	drive_right
country_code				
USCA	US	United States	809.0	False
ASPAC	AUS	Australia	731.0	True
ASPAC	JAP	Japan	588.0	True
ASPAC	IN	India	18.0	True
ASPAC	RU	Russia	200.0	False
LATAM	MOR	Morocco	70.0	False
AFR	EG	Egypt	45.0	False
EUR	ENG	England	NaN	True

Delete the index name

In [51]:
```
cars_df.index.name = None
cars_df
```

Out[51]:

	region	country	cars_per_cap	drive_right
USCA	US	United States	809.0	False
ASPAC	AUS	Australia	731.0	True
ASPAC	JAP	Japan	588.0	True
ASPAC	IN	India	18.0	True
ASPAC	RU	Russia	200.0	False
LATAM	MOR	Morocco	70.0	False
AFR	EG	Egypt	45.0	False
EUR	ENG	England	NaN	True

Set Hierarchical index

In [52]:
```
# Read file and set 1st column as index
cars_df = pd.read_csv("cars.csv", header= None)

# set the column names
cars_df.columns = ['country_code','region','country','cars_per_cap','drives_right']

cars_df.set_index(['region', 'country_code'], inplace=True)
```

In [53]:
```
cars_df
```

Out[53]:

region	country_code	country	cars_per_cap	drives_right
US	**USCA**	United States	809.0	False
AUS	**ASPAC**	Australia	731.0	True
JAP	**ASPAC**	Japan	588.0	True
IN	**ASPAC**	India	18.0	True
RU	**ASPAC**	Russia	200.0	False
MOR	**LATAM**	Morocco	70.0	False
EG	**AFR**	Egypt	45.0	False
ENG	**EUR**	England	NaN	True

In [ ]:
```

```

Example - 5 (Write Data Frame to file)

Write cars_df to cars_to_csv.csv

In [54]:
```
cars_df.to_csv('cars_to_csv.csv')
```

In [ ]:
```
```

Case Study - Sales Data

In [1]:
```
# All imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Sales and Profit data is read in dataframe "sales"

In [9]:
```
# Read file

sales = pd.read_excel('sales.xlsx')
sales
```

Out[9]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12901.51	78476.06
1	Africa	Southern Africa	85	11768.58	51319.50
2	Africa	North Africa	182	21643.08	86698.89
3	Africa	Eastern Africa	110	8013.04	44182.60
4	Africa	Central Africa	103	15606.30	61689.99
5	Asia Pacific	Western Asia	382	-16766.90	124312.24
6	Asia Pacific	Southern Asia	469	67998.76	351806.60
7	Asia Pacific	Southeastern Asia	533	20948.84	329751.38
8	Asia Pacific	Oceania	646	54734.02	408002.98
9	Asia Pacific	Eastern Asia	414	72805.10	315390.77
10	Asia Pacific	Central Asia	37	-2649.76	8190.74
11	Europe	Western Europe	964	82091.27	656637.14
12	Europe	Southern Europe	338	18911.49	215703.93
13	Europe	Northern Europe	367	43237.44	252969.09
14	Europe	Eastern Europe	241	25050.69	108258.93
15	LATAM	South America	496	12377.59	210710.49
16	LATAM	Central America	930	74679.54	461670.28
17	LATAM	Caribbean	288	13529.59	116333.05
18	USCA	Western US	490	44303.65	251991.83
19	USCA	Southern US	255	19991.83	148771.91
20	USCA	Eastern US	443	47462.04	264973.98
21	USCA	Central US	356	33697.43	170416.31
22	USCA	Canada	49	7246.62	26298.81

In [10]:
```
# Read file and set 1st two columns as index
sales = pd.read_excel('sales.xlsx', index_col = [0,1])

sales
```

Out[10]:

Market	Region	No_of_Orders	Profit	Sales
Africa	Western Africa	251	-12901.51	78476.06
	Southern Africa	85	11768.58	51319.50
	North Africa	182	21643.08	86698.89
	Eastern Africa	110	8013.04	44182.60
	Central Africa	103	15606.30	61689.99
Asia Pacific	Western Asia	382	-16766.90	124312.24
	Southern Asia	469	67998.76	351806.60
	Southeastern Asia	533	20948.84	329751.38
	Oceania	646	54734.02	408002.98
	Eastern Asia	414	72805.10	315390.77
	Central Asia	37	-2649.76	8190.74
Europe	Western Europe	964	82091.27	656637.14
	Southern Europe	338	18911.49	215703.93
	Northern Europe	367	43237.44	252969.09
	Eastern Europe	241	25050.69	108258.93
LATAM	South America	496	12377.59	210710.49
	Central America	930	74679.54	461670.28
	Caribbean	288	13529.59	116333.05
USCA	Western US	490	44303.65	251991.83
	Southern US	255	19991.83	148771.91
	Eastern US	443	47462.04	264973.98
	Central US	356	33697.43	170416.31
	Canada	49	7246.62	26298.81

Example - 1

Display first 3 land last 3 rows of the sales dataframe

In [12]:
```
sales.head() # Default - returns top 5 rows
```

Out[12]:

		No_of_Orders	Profit	Sales
Market	**Region**			
Africa	**Western Africa**	251	-12901.51	78476.06
	Southern Africa	85	11768.58	51319.50
	North Africa	182	21643.08	86698.89
	Eastern Africa	110	8013.04	44182.60
	Central Africa	103	15606.30	61689.99

In [13]:
```
sales.head(3)
```

Out[13]:

		No_of_Orders	Profit	Sales
Market	**Region**			
Africa	**Western Africa**	251	-12901.51	78476.06
	Southern Africa	85	11768.58	51319.50
	North Africa	182	21643.08	86698.89

In [14]:
```
sales.tail()
```

Out[14]:

		No_of_Orders	Profit	Sales
Market	**Region**			
USCA	**Western US**	490	44303.65	251991.83
	Southern US	255	19991.83	148771.91
	Eastern US	443	47462.04	264973.98
	Central US	356	33697.43	170416.31
	Canada	49	7246.62	26298.81

In [15]:
```
sales.tail(3)
```

Out[15]:

Market	Region	No_of_Orders	Profit	Sales
USCA	Eastern US	443	47462.04	264973.98
	Central US	356	33697.43	170416.31
	Canada	49	7246.62	26298.81

Example - 2

Display the information about the data stored in data frame

In [16]:
```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 23 entries, ('Africa', 'Western Africa') to ('USCA', 'Canada')
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   No_of_Orders  23 non-null     int64  
 1   Profit        23 non-null     float64
 2   Sales         23 non-null     float64
dtypes: float64(2), int64(1)
memory usage: 932.0+ bytes
```

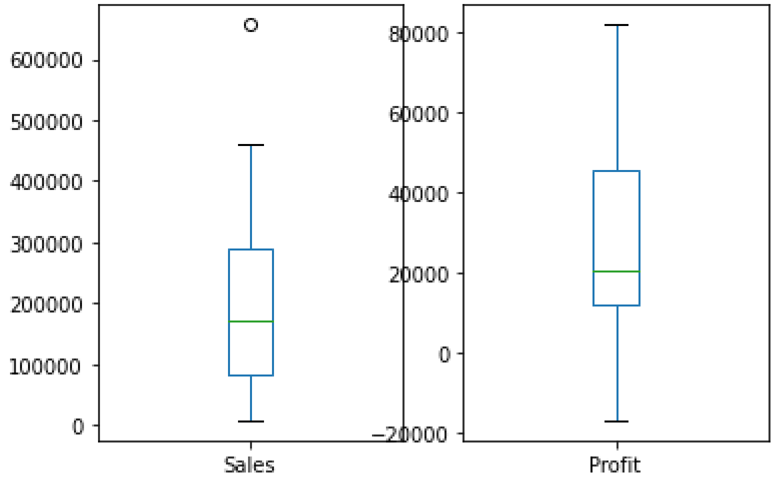
Display the statistical information about the data in dataframe

In [17]:
```
sales.describe()
```

Out[17]:

	No_of_Orders	Profit	Sales
count	23.000000	23.000000	23.000000
mean	366.478261	28859.944783	206285.108696
std	246.590361	27701.193773	160589.886606
min	37.000000	-16766.900000	8190.740000
25%	211.500000	12073.085000	82587.475000
50%	356.000000	20948.840000	170416.310000
75%	479.500000	45882.845000	290182.375000
max	964.000000	82091.270000	656637.140000

In [18]:
```
sales[["Sales", "Profit"]].plot(kind= "box", subplots= True)
plt.show()
```



In [20]:
```
sales["Profit"]
```

Out[20]:
```
Market         Region
Africa         Western Africa      -12901.51
               Southern Africa      11768.58
               North Africa         21643.08
               Eastern Africa        8013.04
               Central Africa       15606.30
Asia Pacific   Western Asia        -16766.90
               Southern Asia        67998.76
               Southeastern Asia    20948.84
               Oceania              54734.02
               Eastern Asia         72805.10
               Central Asia         -2649.76
Europe         Western Europe       82091.27
               Southern Europe      18911.49
               Northern Europe      43237.44
               Eastern Europe       25050.69
LATAM          South America        12377.59
               Central America      74679.54
               Caribbean            13529.59
USCA           Western US           44303.65
               Southern US          19991.83
               Eastern US           47462.04
               Central US           33697.43
               Canada                7246.62
Name: Profit, dtype: float64
```

In [ ]:
```

```

Case Study - Sales Data

In [26]:
```
# All imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Displays pandas float values in 2 decimals
pd.options.display.float_format = '{:,.2f}'.format
```

In [27]:
```
sales = pd.read_excel('sales.xlsx')
sales
```

Out[27]:

	Market	Region	No_of_Orders	Profit	Sales
0	Africa	Western Africa	251	-12,901.51	78,476.06
1	Africa	Southern Africa	85	11,768.58	51,319.50
2	Africa	North Africa	182	21,643.08	86,698.89
3	Africa	Eastern Africa	110	8,013.04	44,182.60
4	Africa	Central Africa	103	15,606.30	61,689.99
5	Asia Pacific	Western Asia	382	-16,766.90	124,312.24
6	Asia Pacific	Southern Asia	469	67,998.76	351,806.60
7	Asia Pacific	Southeastern Asia	533	20,948.84	329,751.38
8	Asia Pacific	Oceania	646	54,734.02	408,002.98
9	Asia Pacific	Eastern Asia	414	72,805.10	315,390.77
10	Asia Pacific	Central Asia	37	-2,649.76	8,190.74
11	Europe	Western Europe	964	82,091.27	656,637.14
12	Europe	Southern Europe	338	18,911.49	215,703.93
13	Europe	Northern Europe	367	43,237.44	252,969.09
14	Europe	Eastern Europe	241	25,050.69	108,258.93
15	LATAM	South America	496	12,377.59	210,710.49
16	LATAM	Central America	930	74,679.54	461,670.28
17	LATAM	Caribbean	288	13,529.59	116,333.05
18	USCA	Western US	490	44,303.65	251,991.83
19	USCA	Southern US	255	19,991.83	148,771.91
20	USCA	Eastern US	443	47,462.04	264,973.98
21	USCA	Central US	356	33,697.43	170,416.31
22	USCA	Canada	49	7,246.62	26,298.81

Sales and Profit data is read in dataframe "sales"

In [28]:
```
# Read file and set 2nd column as index

sales = pd.read_excel('sales.xlsx', index_col = [1])
sales
```

Out[28]:

Region	Market	No_of_Orders	Profit	Sales
Western Africa	Africa	251	-12,901.51	78,476.06
Southern Africa	Africa	85	11,768.58	51,319.50
North Africa	Africa	182	21,643.08	86,698.89
Eastern Africa	Africa	110	8,013.04	44,182.60
Central Africa	Africa	103	15,606.30	61,689.99
Western Asia	Asia Pacific	382	-16,766.90	124,312.24
Southern Asia	Asia Pacific	469	67,998.76	351,806.60
Southeastern Asia	Asia Pacific	533	20,948.84	329,751.38
Oceania	Asia Pacific	646	54,734.02	408,002.98
Eastern Asia	Asia Pacific	414	72,805.10	315,390.77
Central Asia	Asia Pacific	37	-2,649.76	8,190.74
Western Europe	Europe	964	82,091.27	656,637.14
Southern Europe	Europe	338	18,911.49	215,703.93
Northern Europe	Europe	367	43,237.44	252,969.09
Eastern Europe	Europe	241	25,050.69	108,258.93
South America	LATAM	496	12,377.59	210,710.49
Central America	LATAM	930	74,679.54	461,670.28
Caribbean	LATAM	288	13,529.59	116,333.05
Western US	USCA	490	44,303.65	251,991.83
Southern US	USCA	255	19,991.83	148,771.91
Eastern US	USCA	443	47,462.04	264,973.98
Central US	USCA	356	33,697.43	170,416.31
Canada	USCA	49	7,246.62	26,298.81

Example - 1 (Column Indexing)

Display Sales Column

In [10]:
```
sales["Sales"]
```

Out[10]:
```
Region
Western Africa        78,476.06
Southern Africa       51,319.50
North Africa          86,698.89
Eastern Africa        44,182.60
Central Africa        61,689.99
Western Asia         124,312.24
Southern Asia        351,806.60
Southeastern Asia    329,751.38
Oceania              408,002.98
Eastern Asia         315,390.77
Central Asia           8,190.74
Western Europe       656,637.14
Southern Europe      215,703.93
Northern Europe      252,969.09
Eastern Europe       108,258.93
South America        210,710.49
Central America      461,670.28
Caribbean            116,333.05
Western US           251,991.83
Southern US          148,771.91
Eastern US           264,973.98
Central US           170,416.31
Canada                26,298.81
Name: Sales, dtype: float64
```

In [11]:
```
sales.Sales
```

Out[11]:
```
Region
Western Africa        78,476.06
Southern Africa       51,319.50
North Africa          86,698.89
Eastern Africa        44,182.60
Central Africa        61,689.99
Western Asia         124,312.24
Southern Asia        351,806.60
Southeastern Asia    329,751.38
Oceania              408,002.98
Eastern Asia         315,390.77
Central Asia           8,190.74
Western Europe       656,637.14
Southern Europe      215,703.93
Northern Europe      252,969.09
Eastern Europe       108,258.93
South America        210,710.49
Central America      461,670.28
Caribbean            116,333.05
Western US           251,991.83
Southern US          148,771.91
Eastern US           264,973.98
Central US           170,416.31
Canada                26,298.81
Name: Sales, dtype: float64
```

In [12]:
```
type(sales["Sales"])
```

Out[12]:
```
pandas.core.series.Series
```

Display Sales and Profit Column together

In [13]:
```
sales[["Sales", "Profit"]]
```

Out[13]:

Region	Sales	Profit
Western Africa	78,476.06	-12,901.51
Southern Africa	51,319.50	11,768.58
North Africa	86,698.89	21,643.08
Eastern Africa	44,182.60	8,013.04
Central Africa	61,689.99	15,606.30
Western Asia	124,312.24	-16,766.90
Southern Asia	351,806.60	67,998.76
Southeastern Asia	329,751.38	20,948.84
Oceania	408,002.98	54,734.02
Eastern Asia	315,390.77	72,805.10
Central Asia	8,190.74	-2,649.76
Western Europe	656,637.14	82,091.27
Southern Europe	215,703.93	18,911.49
Northern Europe	252,969.09	43,237.44
Eastern Europe	108,258.93	25,050.69
South America	210,710.49	12,377.59
Central America	461,670.28	74,679.54
Caribbean	116,333.05	13,529.59
Western US	251,991.83	44,303.65
Southern US	148,771.91	19,991.83
Eastern US	264,973.98	47,462.04
Central US	170,416.31	33,697.43
Canada	26,298.81	7,246.62

Example - 2 (Row Indexing)

Display data for "Southern Asia"

loc accessor takes row index and column index

In [14]:
```
sales.loc["Southern Asia"]
```

Out[14]:
```
Market              Asia Pacific
No_of_Orders                 469
Profit                 67,998.76
Sales                 351,806.60
Name: Southern Asia, dtype: object
```

Display Sales data for "Southern Asia"

In [15]:
```
sales.loc["Southern Asia", "Sales"]
```

Out[15]:
```
351806.6
```

Display data for "Southern Asia"

iloc accessor takes row number and column number

In [16]:
```
sales.iloc[6]
```

Out[16]:
```
Market              Asia Pacific
No_of_Orders                 469
Profit                 67,998.76
Sales                 351,806.60
Name: Southern Asia, dtype: object
```

In [17]:
```
sales.iloc[6,3]
```

Out[17]:
```
351806.6
```

Example - 3 (Slicing)

Display data for Market, Sales and Profit

In [19]:
```
sales.loc[:, ["Market", "Sales", "Profit"]].head()
```

Out[19]:

Region	Market	Sales	Profit
Western Africa	Africa	78,476.06	-12,901.51
Southern Africa	Africa	51,319.50	11,768.58
North Africa	Africa	86,698.89	21,643.08
Eastern Africa	Africa	44,182.60	8,013.04
Central Africa	Africa	61,689.99	15,606.30

In [20]:
```
sales.iloc[:, [0,3,2] ].head()
```

Out[20]:

Region	Market	Sales	Profit
Western Africa	Africa	78,476.06	-12,901.51
Southern Africa	Africa	51,319.50	11,768.58
North Africa	Africa	86,698.89	21,643.08
Eastern Africa	Africa	44,182.60	8,013.04
Central Africa	Africa	61,689.99	15,606.30

Display data for Western Africa Southern Africa and North Africa

In [21]:
```
sales.loc[["Western Africa", "Southern Africa", "North Africa"] ,:]
```

Out[21]:

Region	Market	No_of_Orders	Profit	Sales
Western Africa	Africa	251	-12,901.51	78,476.06
Southern Africa	Africa	85	11,768.58	51,319.50
North Africa	Africa	182	21,643.08	86,698.89

In [22]:
```
sales.iloc[0:3, :]
```

Out[22]:

Region	Market	No_of_Orders	Profit	Sales
Western Africa	Africa	251	-12,901.51	78,476.06
Southern Africa	Africa	85	11,768.58	51,319.50
North Africa	Africa	182	21,643.08	86,698.89

Display Sales and Profit data for Western Africa Southern Africa and North Africa

In [23]:
```
sales.loc[["Western Africa", "Southern Africa", "North Africa"] , ["Sales", "Profit"]]
```

Out[23]:

Region	Sales	Profit
Western Africa	78,476.06	-12,901.51
Southern Africa	51,319.50	11,768.58
North Africa	86,698.89	21,643.08

In [24]:
```
sales.iloc[0:3, 2:4]
```

Out[24]:

Region	Profit	Sales
Western Africa	-12,901.51	78,476.06
Southern Africa	11,768.58	51,319.50
North Africa	21,643.08	86,698.89

Example - 4 (Filtering)

Display Markets with Sales >300000

In [28]:
```
sales["Sales"] > 300000
```

Out[28]:
```
Region
Western Africa       False
Southern Africa      False
North Africa         False
Eastern Africa       False
Central Africa       False
Western Asia         False
Southern Asia         True
Southeastern Asia     True
Oceania               True
Eastern Asia          True
Central Asia         False
Western Europe        True
Southern Europe      False
Northern Europe      False
Eastern Europe       False
South America        False
Central America       True
Caribbean            False
Western US           False
Southern US          False
Eastern US           False
Central US           False
Canada               False
Name: Sales, dtype: bool
```

In [29]:
```
sales[ sales["Sales"] > 300000 ]
```

Out[29]:

Region	Market	No_of_Orders	Profit	Sales
Southern Asia	Asia Pacific	469	67,998.76	351,806.60
Southeastern Asia	Asia Pacific	533	20,948.84	329,751.38
Oceania	Asia Pacific	646	54,734.02	408,002.98
Eastern Asia	Asia Pacific	414	72,805.10	315,390.77
Western Europe	Europe	964	82,091.27	656,637.14
Central America	LATAM	930	74,679.54	461,670.28

Display the LATAM and Eruopean countries with sales > 250000

In [30]:
```
sales[  (sales["Market"].isin(["LATAM", "Europe"])) & (sales["Sales"] > 250000)     ]
```

Out[30]:

Region	Market	No_of_Orders	Profit	Sales
Western Europe	Europe	964	82,091.27	656,637.14
Northern Europe	Europe	367	43,237.44	252,969.09
Central America	LATAM	930	74,679.54	461,670.28

Optional Examples

The examples given below are good to know but not essential to achieve the objective of this session. You can go through them at your own pace.

Example - 5 (Transformation)

Replace the sales values in the form of thousands

Context: Some time you might want to modify columns to make them more readable. For instance, the sales column in the given data set has six digits, followed by two decimal places. You might want to make it more readable. You can convert the actual sales number to a number in thousands and make it a round figure.

eg. 300000 - 300K

You can use the .floordiv function to achieve the transformation explained above. You can read more about the .floordiv method [here (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.floordiv.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.floordiv.html).

In [18]:
```
sales.Sales = sales.Sales.floordiv(1000)

sales.head()
```

Out[18]:

	Market	No_of_Orders	Profit	Sales
Region				
Western Africa	Africa	251	-12,901.51	78.00
Southern Africa	Africa	85	11,768.58	51.00
North Africa	Africa	182	21,643.08	86.00
Eastern Africa	Africa	110	8,013.04	44.00
Central Africa	Africa	103	15,606.30	61.00

In [19]:
```
sales.rename(columns={'Sales': 'Sales in Thousands'}, inplace=True)
sales.head()
```

Out[19]:

	Market	No_of_Orders	Profit	Sales in Thousands
Region				
Western Africa	Africa	251	-12,901.51	78.00
Southern Africa	Africa	85	11,768.58	51.00
North Africa	Africa	182	21,643.08	86.00
Eastern Africa	Africa	110	8,013.04	44.00
Central Africa	Africa	103	15,606.30	61.00

Replace values in Profit percent of total

In [20]:
```
sales.head()
```

Out[20]:

Region	Market	No_of_Orders	Profit	Sales in Thousands
Western Africa	Africa	251	-12,901.51	78.00
Southern Africa	Africa	85	11,768.58	51.00
North Africa	Africa	182	21,643.08	86.00
Eastern Africa	Africa	110	8,013.04	44.00
Central Africa	Africa	103	15,606.30	61.00

In [21]:
```
#sales['Profit']
total_sum = sales.Profit.sum()
sales['Profit % of Total'] = sales.Profit.apply(lambda x: x/total_sum*100)

sales.head()
```

Out[21]:

Region	Market	No_of_Orders	Profit	Sales in Thousands	Profit % of Total
Western Africa	Africa	251	-12,901.51	78.00	-1.94
Southern Africa	Africa	85	11,768.58	51.00	1.77
North Africa	Africa	182	21,643.08	86.00	3.26
Eastern Africa	Africa	110	8,013.04	44.00	1.21
Central Africa	Africa	103	15,606.30	61.00	2.35

Replace negative Profits with NAN

In [29]:
```
sales.loc[sales['Profit']<0, 'Profit'] = np.nan
sales.head()
```

Out[29]:

Region	Market	No_of_Orders	Profit	Sales
Western Africa	Africa	251	nan	78,476.06
Southern Africa	Africa	85	11,768.58	51,319.50
North Africa	Africa	182	21,643.08	86,698.89
Eastern Africa	Africa	110	8,013.04	44,182.60
Central Africa	Africa	103	15,606.30	61,689.99

Operations on Pandas

This notebook will cover the following topics:

- Filtering dataframes
    - Single and multiple conditions
- Creating new columns
- Lambda functions
- Group by and aggregate functions
- Pivot data
- Merging data frames
    - Joins and concatenations

Preparatory steps

Background

An FMCG company P&J found that the sales of their best selling items are affected by the weather and rainfall trend. For example, the sale of tea increases when it rains, sunscreen is sold on the days when it is least likely to rain, and the sky is clear. They would like to check whether the weather patterns play a vital role in the sale of certain items. Hence as initial experimentation, they would like you to forecast the weather trend in the upcoming days. The target region for this activity is Australia; accordingly, this exercise will be based on analysing and cleaning the weather data from the Australian region available on public platforms.

Read the data into a dataframe

In [1]:
```
import pandas as pd
```

In [2]:
```
data = pd.read_csv("weatherdata.csv", header =0)
```

Display the data

In [3]:
```
data.head(5)
```

Out[3]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0

Data Dictionary

1. Date: The date on which the recording was taken
2. Location: The location of the recording
3. MinTemp: Minimum temperature on the day of the recording (in C)
4. MaxTemp: Maximum temperature in the day of the recording (in C)
5. Rainfall: Rainfall in mm
6. Evaporation: The so-called Class A pan evaporation (mm) in the 24 hours to 9am
7. Sunshine: The number of hours of bright sunshine in the day.
8. WindGustDir: The direction of the strongest wind gust in the 24 hours to midnight
9. WindGustSpeed: The speed (km/h) of the strongest wind gust in the 24 hours to midnight

Example 1.1: Filtering dataframes

Find the days which had sunshine for more that 4 hours. These days will have increased sales of sunscreen.

In [4]:
```
data.shape
```

Out[4]:
```
(142193, 9)
```

In [5]:
```
data["Sunshine"]>4
```

Out[5]:
```
0         False
1         False
2         False
3         False
4         False
5         False
6         False
7         False
8         False
9         False
10        False
11        False
12        False
13        False
14        False
15        False
16        False
17        False
18        False
19        False
20        False
21        False
22        False
23        False
24        False
25        False
26        False
27        False
28        False
29        False
          ...
142163    False
142164    False
142165    False
142166    False
142167    False
142168    False
142169    False
142170    False
142171    False
142172    False
```

```
142173    False
142174    False
142175    False
142176    False
142177    False
142178    False
142179    False
142180    False
142181    False
142182    False
142183    False
142184    False
142185    False
142186    False
142187    False
142188    False
142189    False
142190    False
142191    False
142192    False
Name: Sunshine, Length: 142193, dtype: bool
```

In [6]:
```
data[data["Sunshine"]>4]
```

Out[6]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
5939	2009-01-01	Cobar	17.9	35.2	0.0	12.0	12.3	SSW	48.0
5940	2009-01-02	Cobar	18.4	28.9	0.0	14.8	13.0	S	37.0
5941	2009-01-03	Cobar	15.5	34.1	0.0	12.6	13.3	SE	30.0
5942	2009-01-04	Cobar	19.4	37.6	0.0	10.8	10.6	NNE	46.0
5943	2009-01-05	Cobar	21.9	38.4	0.0	11.4	12.2	WNW	31.0
5944	2009-01-06	Cobar	24.2	41.0	0.0	11.2	8.4	WNW	35.0
5946	2009-01-08	Cobar	23.3	34.0	0.0	9.8	12.6	SSW	41.0
5947	2009-01-09	Cobar	16.1	34.2	0.0	14.6	13.2	SE	37.0
5948	2009-01-10	Cobar	19.0	35.5	0.0	12.0	12.3	ENE	48.0
5949	2009-01-11	Cobar	19.7	35.5	0.0	11.0	12.7	NE	41.0
5950	2009-01-12	Cobar	20.9	37.8	0.0	12.8	13.2	E	30.0
5951	2009-01-13	Cobar	23.9	39.1	0.0	13.8	12.1	ENE	39.0
5952	2009-01-14	Cobar	24.9	41.2	0.0	14.8	13.0	SSW	43.0
5953	2009-01-15	Cobar	25.2	40.5	0.0	16.4	10.3	SW	44.0
5954	2009-01-16	Cobar	21.6	34.2	0.0	17.4	13.1	SW	44.0
5955	2009-01-17	Cobar	18.4	31.8	0.0	16.0	12.9	S	33.0
5956	2009-01-18	Cobar	17.9	34.2	0.0	12.0	11.3	SE	61.0
5957	2009-01-19	Cobar	21.4	37.5	0.0	14.8	6.9	NNE	43.0
5958	2009-01-20	Cobar	23.3	39.4	4.8	12.0	10.9	W	59.0
5960	2009-01-22	Cobar	21.8	30.7	0.0	8.0	5.9	WNW	56.0
5961	2009-01-23	Cobar	20.3	36.0	18.0	8.2	10.5	WSW	94.0
5962	2009-01-24	Cobar	22.1	34.7	8.6	8.6	12.4	NNW	50.0
5963	2009-01-25	Cobar	19.7	37.3	0.0	14.2	13.4	SSW	28.0
5964	2009-01-26	Cobar	23.8	39.9	0.0	12.6	13.2	S	31.0
5965	2009-01-27	Cobar	27.0	38.7	0.0	14.2	13.0	ENE	46.0
5966	2009-01-28	Cobar	26.2	38.5	0.0	14.6	13.3	E	39.0

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
5967	2009-01-29	Cobar	25.0	39.5	0.0	14.6	13.6	ENE	52.0
5968	2009-01-30	Cobar	25.1	39.3	0.0	15.8	13.2	ESE	44.0
5969	2009-01-31	Cobar	25.2	38.5	0.0	16.2	13.1	ENE	44.0
5970	2009-02-01	Cobar	24.8	40.8	0.0	13.4	11.3	SE	30.0
...	...	...	...	...	...	...	...	...	...
139083	2017-05-26	Darwin	24.3	31.9	0.0	6.4	10.9	ESE	50.0
139084	2017-05-27	Darwin	23.5	33.3	0.0	5.4	10.8	E	43.0
139085	2017-05-28	Darwin	22.4	32.2	0.0	8.0	6.3	E	39.0
139086	2017-05-29	Darwin	24.1	32.6	0.0	6.8	10.4	E	52.0
139087	2017-05-30	Darwin	22.5	32.3	0.0	6.6	10.6	SE	48.0
139088	2017-05-31	Darwin	20.4	31.4	0.2	9.6	9.6	ESE	52.0
139089	2017-06-01	Darwin	21.9	31.6	0.0	10.0	9.9	ESE	43.0
139090	2017-06-02	Darwin	21.6	32.0	0.0	9.6	10.7	ESE	43.0
139091	2017-06-03	Darwin	22.7	31.6	0.0	7.0	7.4	ESE	43.0
139092	2017-06-04	Darwin	22.4	31.4	0.0	6.6	8.3	E	43.0
139093	2017-06-05	Darwin	23.3	32.4	0.0	5.4	8.9	E	43.0
139094	2017-06-06	Darwin	20.6	31.8	0.0	6.4	10.8	E	46.0
139095	2017-06-07	Darwin	20.0	30.4	0.0	9.0	10.8	ESE	43.0
139096	2017-06-08	Darwin	19.2	29.4	0.0	7.4	10.9	E	54.0
139097	2017-06-09	Darwin	20.6	29.4	0.0	10.6	5.4	E	46.0
139098	2017-06-10	Darwin	18.7	29.4	0.0	7.8	8.7	ESE	48.0
139099	2017-06-11	Darwin	19.0	29.4	0.0	7.6	10.4	E	46.0
139100	2017-06-12	Darwin	17.2	29.1	0.0	7.2	10.1	ESE	44.0
139101	2017-06-13	Darwin	18.3	29.8	0.0	8.4	10.5	ESE	41.0
139102	2017-06-14	Darwin	16.9	30.3	0.0	5.4	10.9	E	33.0
139103	2017-06-15	Darwin	19.0	30.9	0.0	5.0	10.8	E	41.0

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
139104	2017-06-16	Darwin	18.9	31.1	0.0	6.0	10.6	ESE	31.0
139105	2017-06-17	Darwin	20.2	32.1	0.0	4.8	9.9	E	31.0
139106	2017-06-18	Darwin	20.0	33.1	0.0	4.6	10.9	E	43.0
139107	2017-06-19	Darwin	21.9	33.0	0.0	5.2	10.9	E	44.0
139108	2017-06-20	Darwin	19.3	33.4	0.0	6.0	11.0	ENE	35.0
139109	2017-06-21	Darwin	21.2	32.6	0.0	7.6	8.6	E	37.0
139110	2017-06-22	Darwin	20.7	32.8	0.0	5.6	11.0	E	33.0
139111	2017-06-23	Darwin	19.5	31.8	0.0	6.2	10.6	ESE	26.0
139112	2017-06-24	Darwin	20.2	31.7	0.0	5.6	10.7	ENE	30.0

58898 rows × 9 columns

Note: High sunshine corresponds to low rainfall.

Example 1.2: Filtering dataframes

The cold drink sales will most likely increase on the days which have high sunshine(>5) and high max temperature(>35). Use the filter operation to filter out these days

In [7]:
```
data[(data["MaxTemp"]>35) & (data["Sunshine"]>5)]
```

Out[7]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
5939	2009-01-01	Cobar	17.9	35.2	0.0	12.0	12.3	SSW	48.0
5942	2009-01-04	Cobar	19.4	37.6	0.0	10.8	10.6	NNE	46.0
5943	2009-01-05	Cobar	21.9	38.4	0.0	11.4	12.2	WNW	31.0
5944	2009-01-06	Cobar	24.2	41.0	0.0	11.2	8.4	WNW	35.0
5948	2009-01-10	Cobar	19.0	35.5	0.0	12.0	12.3	ENE	48.0
5949	2009-01-11	Cobar	19.7	35.5	0.0	11.0	12.7	NE	41.0
5950	2009-01-12	Cobar	20.9	37.8	0.0	12.8	13.2	E	30.0
5951	2009-01-13	Cobar	23.9	39.1	0.0	13.8	12.1	ENE	39.0
5952	2009-01-14	Cobar	24.9	41.2	0.0	14.8	13.0	SSW	43.0
5953	2009-01-15	Cobar	25.2	40.5	0.0	16.4	10.3	SW	44.0
5957	2009-01-19	Cobar	21.4	37.5	0.0	14.8	6.9	NNE	43.0
5958	2009-01-20	Cobar	23.3	39.4	4.8	12.0	10.9	W	59.0
5961	2009-01-23	Cobar	20.3	36.0	18.0	8.2	10.5	WSW	94.0
5963	2009-01-25	Cobar	19.7	37.3	0.0	14.2	13.4	SSW	28.0
5964	2009-01-26	Cobar	23.8	39.9	0.0	12.6	13.2	S	31.0
5965	2009-01-27	Cobar	27.0	38.7	0.0	14.2	13.0	ENE	46.0
5966	2009-01-28	Cobar	26.2	38.5	0.0	14.6	13.3	E	39.0
5967	2009-01-29	Cobar	25.0	39.5	0.0	14.6	13.6	ENE	52.0
5968	2009-01-30	Cobar	25.1	39.3	0.0	15.8	13.2	ESE	44.0
5969	2009-01-31	Cobar	25.2	38.5	0.0	16.2	13.1	ENE	44.0
5970	2009-02-01	Cobar	24.8	40.8	0.0	13.4	11.3	SE	30.0
5971	2009-02-02	Cobar	27.6	40.3	0.0	14.4	10.9	S	57.0
5972	2009-02-03	Cobar	23.6	40.4	0.6	11.8	12.2	WSW	54.0
5973	2009-02-04	Cobar	24.1	41.4	1.6	12.6	12.3	ENE	39.0
5974	2009-02-05	Cobar	27.2	43.4	0.0	14.2	12.6	NNW	37.0
5975	2009-02-06	Cobar	29.1	43.5	0.0	13.0	12.1	WNW	28.0

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
5976	2009-02-07	Cobar	28.9	41.4	0.0	15.6	12.7	NNE	41.0
5977	2009-02-08	Cobar	25.1	42.0	0.0	17.4	13.0	NNE	39.0
5978	2009-02-09	Cobar	25.4	36.6	0.0	15.2	10.3	SW	43.0
5992	2009-02-23	Cobar	21.9	35.1	0.0	9.0	10.2	S	43.0
...	...	...	...	...	...	...	...	...	...
138541	2015-12-01	Darwin	25.1	35.5	1.2	5.2	7.8	N	28.0
138543	2015-12-03	Darwin	25.6	35.6	0.0	5.0	11.0	ESE	37.0
138546	2015-12-06	Darwin	25.6	35.9	0.0	4.6	12.0	N	41.0
138547	2015-12-07	Darwin	27.0	36.4	0.0	8.4	11.9	NNE	33.0
138584	2016-01-13	Darwin	25.2	35.5	0.0	8.0	7.8	S	39.0
138623	2016-02-21	Darwin	27.4	35.3	0.0	7.0	9.9	NNW	30.0
138632	2016-03-01	Darwin	26.6	35.2	0.0	8.0	11.5	NW	28.0
138645	2016-03-14	Darwin	26.2	35.1	1.6	6.0	6.6	WSW	44.0
138673	2016-04-11	Darwin	25.7	35.9	0.0	9.2	11.2	E	39.0
138674	2016-04-12	Darwin	25.6	36.0	0.0	6.8	10.7	E	43.0
138675	2016-04-13	Darwin	26.1	35.7	0.0	9.8	9.5	ENE	44.0
138683	2016-04-21	Darwin	24.9	35.7	0.0	5.2	11.1	E	41.0
138684	2016-04-22	Darwin	24.9	35.9	0.0	6.0	11.0	E	41.0
138685	2016-04-23	Darwin	24.9	35.5	0.0	8.0	9.8	E	52.0
138686	2016-04-24	Darwin	23.9	35.4	0.0	8.0	10.8	SSE	56.0
138687	2016-04-25	Darwin	23.7	35.6	0.0	9.0	11.1	ESE	57.0
138688	2016-04-26	Darwin	23.3	35.4	0.0	8.0	10.0	E	33.0
138709	2016-05-17	Darwin	24.8	35.1	0.0	6.6	10.9	ENE	39.0
138710	2016-05-18	Darwin	25.0	35.2	0.0	3.8	11.0	ESE	33.0
138715	2016-05-23	Darwin	25.9	35.1	0.0	4.8	11.1	S	37.0
138821	2016-09-06	Darwin	24.0	35.2	0.0	8.0	10.7	NW	41.0

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed
138837	2016-09-22	Darwin	25.0	35.5	0.0	6.8	9.3	NNE	39.0
138851	2016-10-06	Darwin	24.8	35.8	0.0	8.0	9.3	E	41.0
138852	2016-10-07	Darwin	26.3	36.2	0.0	8.0	11.0	E	44.0
138853	2016-10-08	Darwin	25.7	37.5	0.0	8.0	11.5	E	50.0
138862	2016-10-17	Darwin	25.1	35.2	0.0	7.4	11.5	NNE	39.0
138879	2016-11-03	Darwin	24.4	35.5	0.0	7.8	9.9	NW	35.0
138892	2016-11-16	Darwin	25.7	35.2	0.0	5.4	11.3	NW	26.0
138905	2016-11-29	Darwin	25.8	35.1	0.8	4.8	6.4	SSE	46.0
138910	2016-12-04	Darwin	25.8	35.2	0.0	NaN	12.0	ESE	33.0

3861 rows × 9 columns

Note: The construction of the filter condition, it has individual filter conditions separated in parenthesis

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Example 2.1: Creating new columns

If you noticed the filtering done in the earlier examples did not give precise information about the days, the data column simply has the dates. The date column can be split into the year, month and day of the month.

Special module of pandas The "DatetimeIndex" is a particular module which has the capabilities to extract a day, month and year form the date.

In [8]:
```
pd.DatetimeIndex(data["Date"])
```

Out[8]:
```
DatetimeIndex(['2008-12-01', '2008-12-02', '2008-12-03', '2008-12-04',
               '2008-12-05', '2008-12-06', '2008-12-07', '2008-12-08',
               '2008-12-09', '2008-12-10',
               ...
               '2017-06-15', '2017-06-16', '2017-06-17', '2017-06-18',
               '2017-06-19', '2017-06-20', '2017-06-21', '2017-06-22',
               '2017-06-23', '2017-06-24'],
              dtype='datetime64[ns]', name='Date', length=142193, freq=None)
```

In [9]:
```
pd.DatetimeIndex(data["Date"]).year
```

Out[9]:
```
Int64Index([2008, 2008, 2008, 2008, 2008, 2008, 2008, 2008, 2008, 2008,
            ...
            2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017],
           dtype='int64', name='Date', length=142193)
```

Adding New columns To add a new column in the dataframe just name the column and pass the instructions about the creation of the new column

In [10]:
```
data["Year"] = pd.DatetimeIndex(data["Date"]).year
```

In [11]:
```
data.head()
```

Out[11]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	2008
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	2008
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	2008
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	2008
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	2008

In [12]:
```
data["Month"] = pd.DatetimeIndex(data["Date"]).month
```

In [13]:
```
data["Dayofmonth"] = pd.DatetimeIndex(data["Date"]).day
```

In [14]:
```
data.head(20)
```

Out[14]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	2008	12	1
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	2008	12	2
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	2008	12	3
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	2008	12	4
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	2008	12	5
5	2008-12-06	Albury	14.6	29.7	0.2	NaN	NaN	WNW	56.0	2008	12	6
6	2008-12-07	Albury	14.3	25.0	0.0	NaN	NaN	W	50.0	2008	12	7
7	2008-12-08	Albury	7.7	26.7	0.0	NaN	NaN	W	35.0	2008	12	8
8	2008-12-09	Albury	9.7	31.9	0.0	NaN	NaN	NNW	80.0	2008	12	9
9	2008-12-10	Albury	13.1	30.1	1.4	NaN	NaN	W	28.0	2008	12	10
10	2008-12-11	Albury	13.4	30.4	0.0	NaN	NaN	N	30.0	2008	12	11
11	2008-12-12	Albury	15.9	21.7	2.2	NaN	NaN	NNE	31.0	2008	12	12
12	2008-12-13	Albury	15.9	18.6	15.6	NaN	NaN	W	61.0	2008	12	13
13	2008-12-14	Albury	12.6	21.0	3.6	NaN	NaN	SW	44.0	2008	12	14
14	2008-12-16	Albury	9.8	27.7	NaN	NaN	NaN	WNW	50.0	2008	12	16
15	2008-12-17	Albury	14.1	20.9	0.0	NaN	NaN	ENE	22.0	2008	12	17
16	2008-12-18	Albury	13.5	22.9	16.8	NaN	NaN	W	63.0	2008	12	18
17	2008-12-19	Albury	11.2	22.5	10.6	NaN	NaN	SSE	43.0	2008	12	19
18	2008-12-20	Albury	9.8	25.6	0.0	NaN	NaN	SSE	26.0	2008	12	20
19	2008-12-21	Albury	11.5	29.3	0.0	NaN	NaN	S	24.0	2008	12	21

Example 2.2: Creating new columns

The temperature given is in Celcius, convert it in Fahrenheit and store it in a new column for it.

In [15]:
```
data["Maxtemp_F"] = data["MaxTemp"] * 9/5 +32
```

In [16]:
```
data.head()
```

Out[16]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_F
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	2008	12	1	73.22
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	2008	12	2	77.18
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	2008	12	3	78.26
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	2008	12	4	82.40
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	2008	12	5	90.14

Example 3.1: Lambda Functions

Let's create a new column which highlights the days which have rainfall more than 50 mm as rainy days and the rest are not.

In [18]:
```
data.Rainfall
```

Out[18]:
```
0          0.6
1          0.0
2          0.0
3          0.0
4          1.0
5          0.2
6          0.0
7          0.0
8          0.0
9          1.4
10         0.0
11         2.2
12        15.6
13         3.6
14         NaN
15         0.0
16        16.8
17        10.6
18         0.0
19         0.0
20         0.0
21         0.0
22         0.0
23         0.0
24         0.0
25         0.0
26         0.0
27         0.0
28         1.2
29         0.8
          ...
142163     0.4
142164     0.0
142165     0.0
142166     0.0
142167     0.0
142168     0.0
142169     0.0
142170     0.0
142171     0.0
142172     0.0
```

```
142173     0.0
142174     0.0
142175     0.0
142176     0.0
142177     0.0
142178     0.0
142179     0.0
142180     0.0
142181     0.0
142182     0.0
142183     0.0
142184     0.0
142185     0.0
142186     0.0
142187     0.0
142188     0.0
142189     0.0
142190     0.0
142191     0.0
142192     0.0
Name: Rainfall, Length: 142193, dtype: float64
```

In [17]:
```
data.Rainfall.apply(lambda x: "Rainy" if x > 50  else "Not rainy")
```

Out[17]:
```
0         Not rainy
1         Not rainy
2         Not rainy
3         Not rainy
4         Not rainy
5         Not rainy
6         Not rainy
7         Not rainy
8         Not rainy
9         Not rainy
10        Not rainy
11        Not rainy
12        Not rainy
13        Not rainy
14        Not rainy
15        Not rainy
16        Not rainy
17        Not rainy
18        Not rainy
19        Not rainy
20        Not rainy
21        Not rainy
22        Not rainy
23        Not rainy
24        Not rainy
25        Not rainy
26        Not rainy
27        Not rainy
28        Not rainy
29        Not rainy
            ...
142163    Not rainy
142164    Not rainy
142165    Not rainy
142166    Not rainy
142167    Not rainy
142168    Not rainy
142169    Not rainy
142170    Not rainy
142171    Not rainy
142172    Not rainy
```

```
142173    Not rainy
142174    Not rainy
142175    Not rainy
142176    Not rainy
142177    Not rainy
142178    Not rainy
142179    Not rainy
142180    Not rainy
142181    Not rainy
142182    Not rainy
142183    Not rainy
142184    Not rainy
142185    Not rainy
142186    Not rainy
142187    Not rainy
142188    Not rainy
142189    Not rainy
142190    Not rainy
142191    Not rainy
142192    Not rainy
Name: Rainfall, Length: 142193, dtype: object
```

Note

1. New way of accessing a column in a dataframe by using the dot operator.
2. "apply" function takes in a lambda operator as argument.

In [19]:
```
type(data.Rainfall)
```

Out[19]:
```
pandas.core.series.Series
```

In [20]:
```
type(data["Rainfall"])
```

Out[20]:
```
pandas.core.series.Series
```

In [21]:
```
data["is_raining"] = data.Rainfall.apply(lambda x: "Rainy" if x > 50  else "Not rainy")
```

In [24]:
```
## Note that the above code is also another way to find this
## data["is_raining"] = data[Rainfall]apply(lambda x: "Rainy" if x > 50  else "Not rainy")
```

In [22]:
```
data[data["is_raining"] == "Rainy"]
```

Out[22]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth
429	2010-02-05	Albury	19.2	26.1	52.2	NaN	NaN	SE	33.0	2010	2	5
455	2010-03-08	Albury	18.1	25.5	66.0	NaN	NaN	NW	56.0	2010	3	8
690	2010-10-31	Albury	13.8	18.7	50.8	NaN	NaN	NNW	52.0	2010	10	31
704	2010-11-14	Albury	19.2	22.6	52.6	NaN	NaN	N	26.0	2010	11	14
787	2011-02-05	Albury	20.4	23.0	99.2	NaN	NaN	NW	28.0	2011	2	5
788	2011-02-06	Albury	14.7	21.5	51.0	NaN	NaN	WSW	43.0	2011	2	6
1142	2012-03-01	Albury	17.1	20.9	104.2	NaN	NaN	SE	57.0	2012	3	1

In [ ]:

In [ ]:

In [ ]:

Example 4.1: Grouping and Aggregate functions

Find the location which received the most amount of rain in the given data. In this place, certain promotional offers can be put in place to boost

In [25]:
```
data.head()
```

Out[25]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_F	is
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	2008	12	1	73.22	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	2008	12	2	77.18	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	2008	12	3	78.26	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	2008	12	4	82.40	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	2008	12	5	90.14	

In [32]:
```
data.tail()
```

Out[32]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp
142188	2017-06-20	Uluru	3.5	21.8	0.0	NaN	NaN	E	31.0	2017	6	20	71.
142189	2017-06-21	Uluru	2.8	23.4	0.0	NaN	NaN	E	31.0	2017	6	21	74.
142190	2017-06-22	Uluru	3.6	25.3	0.0	NaN	NaN	NNW	22.0	2017	6	22	77.
142191	2017-06-23	Uluru	5.4	26.9	0.0	NaN	NaN	N	37.0	2017	6	23	80.
142192	2017-06-24	Uluru	7.8	27.0	0.0	NaN	NaN	SE	28.0	2017	6	24	80.

In [29]:
```
data_bylocation = data.groupby(by = ['Location']).mean()
data_bylocation.head()
```

Out[29]:

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_F
Adelaide	12.628368	22.945402	1.572185	5.824924	7.752002	36.530812	2012.525890	6.523948	15.740453	73.301723
Albany	12.948461	20.072587	2.255073	4.207273	6.658765	NaN	2012.708554	6.413130	15.680371	68.130657
Albury	9.520899	22.630963	1.925710	NaN	NaN	32.953016	2012.733643	6.412488	15.745932	72.735734
AliceSprings	13.125182	29.244191	0.869355	9.029929	9.581944	40.533714	2012.719565	6.407456	15.689211	84.639545
BadgerysCreek	11.136900	24.023111	2.207925	NaN	NaN	33.609890	2012.790984	6.326161	15.769467	75.241600

In [31]:
```
data_bylocation.sort_values('Rainfall', ascending = False).head()
```

Out[31]:

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_F
Cairns	21.199197	29.544344	5.765317	6.211976	7.575995	38.067991	2012.677376	6.363454	15.720214	85.179819
Darwin	23.210530	32.540977	5.094048	6.319089	8.499310	40.582355	2012.502820	6.534461	15.716792	90.573759
CoffsHarbour	14.365774	23.915575	5.054592	3.904267	7.362374	39.232197	2012.749746	6.392482	15.716898	75.048035
GoldCoast	17.341490	25.752971	3.728933	NaN	NaN	42.472539	2012.683221	6.435906	15.717114	78.355347
Wollongong	14.949058	21.476510	3.589127	NaN	NaN	45.695257	2012.743882	6.423734	15.694268	70.657718

In [ ]:

In [ ]:

Example 4.2: Grouping and Aggregate functions

Hot chocolate is the most sold product in the cold months. Find month which is the coldest so that the inventory team can keep the stock of hot chocolate ready well in advance.

In [42]:
```
data_bymonth = data.groupby(by = ['Month']).mean()
data_bymonth
```

Out[42]:

Month	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	Year	Dayofmonth	Maxtemp_F	WCI
1	17.520778	29.547362	2.719036	8.773171	9.208942	43.361730	2013.042721	15.986688	85.185252	504.169996
2	17.500239	28.877704	3.174075	7.651018	8.607494	41.457472	2013.054822	14.643515	83.979867	511.722359
3	15.904347	26.886744	2.801304	6.237989	7.646279	39.546399	2013.024778	15.995321	80.396138	570.372892
4	12.831979	23.611845	2.314764	4.547511	7.107208	36.460285	2013.279055	15.492659	74.501320	680.791840
5	9.618572	20.047202	1.978896	3.244134	6.337496	35.721056	2013.040214	15.991038	68.084964	787.434259
6	7.815031	17.324778	2.781114	2.518705	5.660379	35.506375	2012.975381	15.257648	63.184600	845.755217
7	6.951308	16.764242	2.179314	2.699269	6.069790	37.891458	2012.467867	16.001528	62.175636	863.519699
8	7.465145	18.258930	2.029610	3.616533	7.171661	40.245052	2012.473474	16.022275	64.866074	836.501471
9	9.460189	20.772510	1.875851	4.917265	7.698770	42.213311	2012.461084	15.518378	69.390517	762.816683
10	11.531145	23.540695	1.610734	6.379571	8.500080	42.716694	2012.462725	16.026771	74.373252	697.875616
11	14.299624	26.165571	2.273758	7.465236	8.685394	42.582385	2012.435041	15.498211	79.098028	612.435126
12	15.771514	27.526390	2.476483	8.046298	8.975372	43.004769	2012.286401	15.969103	81.547503	561.241935

In [35]:
```
data_bymonth.sort_values('MinTemp')
```

Out[35]:

Month	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	Year	Dayofmonth	Maxtemp_F
7	6.951308	16.764242	2.179314	2.699269	6.069790	37.891458	2012.467867	16.001528	62.175636
8	7.465145	18.258930	2.029610	3.616533	7.171661	40.245052	2012.473474	16.022275	64.866074
6	7.815031	17.324778	2.781114	2.518705	5.660379	35.506375	2012.975381	15.257648	63.184600
9	9.460189	20.772510	1.875851	4.917265	7.698770	42.213311	2012.461084	15.518378	69.390517
5	9.618572	20.047202	1.978896	3.244134	6.337496	35.721056	2013.040214	15.991038	68.084964
10	11.531145	23.540695	1.610734	6.379571	8.500080	42.716694	2012.462725	16.026771	74.373252
4	12.831979	23.611845	2.314764	4.547511	7.107208	36.460285	2013.279055	15.492659	74.501320
11	14.299624	26.165571	2.273758	7.465236	8.685394	42.582385	2012.435041	15.498211	79.098028
12	15.771514	27.526390	2.476483	8.046298	8.975372	43.004769	2012.286401	15.969103	81.547503
3	15.904347	26.886744	2.801304	6.237989	7.646279	39.546399	2013.024778	15.995321	80.396138
2	17.500239	28.877704	3.174075	7.651018	8.607494	41.457472	2013.054822	14.643515	83.979867

Example 4.3: Grouping and Aggregate functions

Sometimes feeling cold is more than about low temperatures; a windy day can also make you cold. A factor called the chill factor can be used to quantify the cold based on the wind speed and the temperature. The formula for the chill factor is given by

$$WCI = (10 * \sqrt{v} - v + 10.5).(33 - T_m)$$

v is the speed of the wind and T_m is the minimum temperature

Add a column for WCI and find the month with the lowest WCI.

In [37]:
```
from math import sqrt
def wci(x):
    velocity = x['WindGustSpeed']
    minTemp = x['MinTemp']
    return ((10 * sqrt(velocity) - velocity + 10.5)*(33-minTemp))
```

In [38]:
```
data['WCI'] = data.apply(wci,axis=1)
```

In [39]:
```
data.head()
```

Out[39]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_F	is
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	2008	12	1	73.22	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	2008	12	2	77.18	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	2008	12	3	78.26	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	2008	12	4	82.40	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	2008	12	5	90.14	

In [41]:

```
data_bymonth = data.groupby(by = ['Month']).mean()
data_bymonth
```

Out[41]:

Month	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	Year	Dayofmonth	Maxtemp_F	WCI
1	17.520778	29.547362	2.719036	8.773171	9.208942	43.361730	2013.042721	15.986688	85.185252	504.169996
2	17.500239	28.877704	3.174075	7.651018	8.607494	41.457472	2013.054822	14.643515	83.979867	511.722359
3	15.904347	26.886744	2.801304	6.237989	7.646279	39.546399	2013.024778	15.995321	80.396138	570.372892
4	12.831979	23.611845	2.314764	4.547511	7.107208	36.460285	2013.279055	15.492659	74.501320	680.791840
5	9.618572	20.047202	1.978896	3.244134	6.337496	35.721056	2013.040214	15.991038	68.084964	787.434259
6	7.815031	17.324778	2.781114	2.518705	5.660379	35.506375	2012.975381	15.257648	63.184600	845.755217
7	6.951308	16.764242	2.179314	2.699269	6.069790	37.891458	2012.467867	16.001528	62.175636	863.519699
8	7.465145	18.258930	2.029610	3.616533	7.171661	40.245052	2012.473474	16.022275	64.866074	836.501471
9	9.460189	20.772510	1.875851	4.917265	7.698770	42.213311	2012.461084	15.518378	69.390517	762.816683
10	11.531145	23.540695	1.610734	6.379571	8.500080	42.716694	2012.462725	16.026771	74.373252	697.875616
11	14.299624	26.165571	2.273758	7.465236	8.685394	42.582385	2012.435041	15.498211	79.098028	612.435126
12	15.771514	27.526390	2.476483	8.046298	8.975372	43.004769	2012.286401	15.969103	81.547503	561.241935

In [44]:
```
data_bymonth.sort_values('WCI', ascending = False)
```

Out[44]:

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	Year	Dayofmonth	Maxtemp_F	WCI
Month										
7	6.951308	16.764242	2.179314	2.699269	6.069790	37.891458	2012.467867	16.001528	62.175636	863.519699
6	7.815031	17.324778	2.781114	2.518705	5.660379	35.506375	2012.975381	15.257648	63.184600	845.755217
8	7.465145	18.258930	2.029610	3.616533	7.171661	40.245052	2012.473474	16.022275	64.866074	836.501471
5	9.618572	20.047202	1.978896	3.244134	6.337496	35.721056	2013.040214	15.991038	68.084964	787.434259
9	9.460189	20.772510	1.875851	4.917265	7.698770	42.213311	2012.461084	15.518378	69.390517	762.816683
10	11.531145	23.540695	1.610734	6.379571	8.500080	42.716694	2012.462725	16.026771	74.373252	697.875616
4	12.831979	23.611845	2.314764	4.547511	7.107208	36.460285	2013.279055	15.492659	74.501320	680.791840
11	14.299624	26.165571	2.273758	7.465236	8.685394	42.582385	2012.435041	15.498211	79.098028	612.435126
3	15.904347	26.886744	2.801304	6.237989	7.646279	39.546399	2013.024778	15.995321	80.396138	570.372892
12	15.771514	27.526390	2.476483	8.046298	8.975372	43.004769	2012.286401	15.969103	81.547503	561.241935
2	17.500239	28.877704	3.174075	7.651018	8.607494	41.457472	2013.054822	14.643515	83.979867	511.722359
1	17.520778	29.547362	2.719036	8.773171	9.208942	43.361730	2013.042721	15.986688	85.185252	504.169996

In [43]:
```
#The month with the lowest WCI
data_bymonth.sort_values('WCI')
```

Out[43]:

Month	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	Year	Dayofmonth	Maxtemp_F	WCI
1	17.520778	29.547362	2.719036	8.773171	9.208942	43.361730	2013.042721	15.986688	85.185252	504.169996
2	17.500239	28.877704	3.174075	7.651018	8.607494	41.457472	2013.054822	14.643515	83.979867	511.722359
12	15.771514	27.526390	2.476483	8.046298	8.975372	43.004769	2012.286401	15.969103	81.547503	561.241935
3	15.904347	26.886744	2.801304	6.237989	7.646279	39.546399	2013.024778	15.995321	80.396138	570.372892
11	14.299624	26.165571	2.273758	7.465236	8.685394	42.582385	2012.435041	15.498211	79.098028	612.435126
4	12.831979	23.611845	2.314764	4.547511	7.107208	36.460285	2013.279055	15.492659	74.501320	680.791840
10	11.531145	23.540695	1.610734	6.379571	8.500080	42.716694	2012.462725	16.026771	74.373252	697.875616
9	9.460189	20.772510	1.875851	4.917265	7.698770	42.213311	2012.461084	15.518378	69.390517	762.816683
5	9.618572	20.047202	1.978896	3.244134	6.337496	35.721056	2013.040214	15.991038	68.084964	787.434259
8	7.465145	18.258930	2.029610	3.616533	7.171661	40.245052	2012.473474	16.022275	64.866074	836.501471
6	7.815031	17.324778	2.781114	2.518705	5.660379	35.506375	2012.975381	15.257648	63.184600	845.755217
7	6.951308	16.764242	2.179314	2.699269	6.069790	37.891458	2012.467867	16.001528	62.175636	863.519699

Example 5.1: Merging Dataframes

The join command is used to combine dataframes. Unlike hstack and vstack, the join command works by using a key to combine to dataframes.

For example the total tea for the Newcastle store for the month of June 2011 is given in the file names `junesales.csv` Read in the data from the file and join it to the weather data exracted from the original dataframe.

In [45]:
```
sales = pd.read_csv("junesales.csv", header = 0)
```

In [46]:
```
sales["Dayofmonth"] = pd.DatetimeIndex(sales["Date"]).day
sales.head()
```

Out[46]:

	Date	Tea_sales(in 100's)	Dayofmonth
0	6/1/2011	26	1
1	6/2/2011	35	2
2	6/3/2011	37	3
3	6/4/2011	33	4
4	6/5/2011	25	5

In [47]:
```
# Filter the sales data for the relevant month and the appropriate location to a new dataframe.

Newcastle_data = data[(data['Location']=='Newcastle') & (data['Year']==2011) & (data['Month']==6)]
Newcastle_data.head()
```

Out[47]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_
15605	2011-06-01	Newcastle	NaN	21.2	6.0	NaN	NaN	NaN	NaN	2011	6	1	70.
15606	2011-06-02	Newcastle	NaN	20.2	4.0	NaN	NaN	NaN	NaN	2011	6	2	68.
15607	2011-06-03	Newcastle	10.7	20.2	0.4	NaN	NaN	NaN	NaN	2011	6	3	68.
15608	2011-06-04	Newcastle	9.4	20.4	0.0	NaN	NaN	NaN	NaN	2011	6	4	68.
15609	2011-06-05	Newcastle	9.6	18.8	3.0	NaN	NaN	NaN	NaN	2011	6	5	65.

In [48]:
```
merge_data = Newcastle_data.merge(sales, on = "Dayofmonth")
merge_data.head(30)
```

Out[48]:

	Date_x	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_F
0	2011-06-01	Newcastle	NaN	21.2	6.0	NaN	NaN	NaN	NaN	2011	6	1	70.16
1	2011-06-02	Newcastle	NaN	20.2	4.0	NaN	NaN	NaN	NaN	2011	6	2	68.36
2	2011-06-03	Newcastle	10.7	20.2	0.4	NaN	NaN	NaN	NaN	2011	6	3	68.36
3	2011-06-04	Newcastle	9.4	20.4	0.0	NaN	NaN	NaN	NaN	2011	6	4	68.72
4	2011-06-05	Newcastle	9.6	18.8	3.0	NaN	NaN	NaN	NaN	2011	6	5	65.84
5	2011-06-06	Newcastle	8.2	19.5	0.0	NaN	NaN	NaN	NaN	2011	6	6	67.10
6	2011-06-07	Newcastle	5.6	16.7	0.0	NaN	NaN	NaN	NaN	2011	6	7	62.06
7	2011-06-08	Newcastle	8.7	15.2	0.0	NaN	NaN	NaN	NaN	2011	6	8	59.36
8	2011-06-09	Newcastle	5.5	15.6	0.0	NaN	NaN	NaN	NaN	2011	6	9	60.08
9	2011-06-10	Newcastle	7.3	17.2	0.0	NaN	NaN	NaN	NaN	2011	6	10	62.96
10	2011-06-11	Newcastle	10.1	17.0	4.0	NaN	NaN	NaN	NaN	2011	6	11	62.60
11	2011-06-12	Newcastle	10.0	18.3	36.4	NaN	NaN	NaN	NaN	2011	6	12	64.94
12	2011-06-13	Newcastle	NaN	18.5	35.6	NaN	NaN	NaN	NaN	2011	6	13	65.30
13	2011-06-14	Newcastle	NaN	18.7	26.5	NaN	NaN	NaN	NaN	2011	6	14	65.66
14	2011-06-15	Newcastle	NaN	18.5	12.4	NaN	NaN	NaN	NaN	2011	6	15	65.30
15	2011-06-16	Newcastle	NaN	18.6	21.8	NaN	NaN	NaN	NaN	2011	6	16	65.48

	Date_x	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxtemp_F
16	2011-06-17	Newcastle	8.4	18.5	0.4	NaN	NaN	NaN	NaN	2011	6	17	65.30
17	2011-06-18	Newcastle	6.7	17.7	0.0	NaN	NaN	NaN	NaN	2011	6	18	63.86
18	2011-06-19	Newcastle	6.5	18.1	0.0	NaN	NaN	NaN	NaN	2011	6	19	64.58
19	2011-06-20	Newcastle	6.9	18.8	0.0	NaN	NaN	NaN	NaN	2011	6	20	65.84
20	2011-06-21	Newcastle	6.2	19.9	0.0	NaN	NaN	NaN	NaN	2011	6	21	67.82
21	2011-06-22	Newcastle	6.8	15.7	0.8	NaN	NaN	NaN	NaN	2011	6	22	60.26
22	2011-06-23	Newcastle	7.5	18.2	0.0	NaN	NaN	NaN	NaN	2011	6	23	64.76
23	2011-06-24	Newcastle	5.6	18.2	0.0	NaN	NaN	NaN	NaN	2011	6	24	64.76
24	2011-06-25	Newcastle	4.9	18.8	0.0	NaN	NaN	NaN	NaN	2011	6	25	65.84
25	2011-06-26	Newcastle	6.0	20.0	0.0	NaN	NaN	NaN	NaN	2011	6	26	68.00
26	2011-06-27	Newcastle	6.3	20.0	0.0	NaN	NaN	NaN	NaN	2011	6	27	68.00
27	2011-06-28	Newcastle	10.0	18.0	0.4	NaN	NaN	NaN	NaN	2011	6	28	64.40
28	2011-06-29	Newcastle	12.4	19.6	0.0	NaN	NaN	NaN	NaN	2011	6	29	67.28
29	2011-06-30	Newcastle	13.4	17.2	0.6	NaN	NaN	NaN	NaN	2011	6	30	62.96

Example 5.2: Merging Dataframes

Types of joins.

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Each state may have different tax laws, so we might want to add the states information to the data as well.

The file `locationsandstates.csv` information about the states and location, the data in this file is **not** same as the weather data. It is possible that few locations in "data" (original dataframe) are not in this file, and all the locations in the file might not be in the original dataframe.

In the original dataframe add the state data.

In [49]:
```
state = pd.read_csv("locationsandstates.csv", header = 0)
state
```

Out[49]:

	Location	State
0	Sydney	New South Wales
1	Albury	New South Wales
2	Armidale	New South Wales
3	Bathurst	New South Wales
4	Blue Mountains	New South Wales
5	Broken Hill	New South Wales
6	Campbelltown	New South Wales
7	Cessnock	New South Wales
8	Dubbo	New South Wales
9	Goulburn	New South Wales
10	Grafton	New South Wales
11	Lithgow	New South Wales
12	Liverpool	New South Wales
13	Newcastle	New South Wales
14	Orange	New South Wales
15	Parramatta	New South Wales
16	Penrith	New South Wales
17	Queanbeyan	New South Wales
18	Tamworth	New South Wales
19	WaggaWagga	New South Wales
20	Wollongong	New South Wales
21	Darwin	Northern Territory
22	Palmerston	Northern Territory
23	Brisbane	Queensland
24	Bundaberg	Queensland
25	Caboolture	Queensland

	Location	State
26	Cairns	Queensland
27	Caloundra	Queensland
28	Gladstone	Queensland
29	Gold Coast	Queensland
...	...	...
46	Ararat	Victoria
47	Bairnsdale	Victoria
48	Benalla	Victoria
49	Ballarat	Victoria
50	Bendigo	Victoria
51	Dandenong	Victoria
52	Frankston	Victoria
53	Geelong	Victoria
54	Hamilton	Victoria
55	Horsham	Victoria
56	Latrobe City	Victoria
57	Melton	Victoria
58	Mildura	Victoria
59	Sale	Victoria
60	Shepparton	Victoria
61	Swan Hill	Victoria
62	Wangaratta	Victoria
63	Warrnambool	Victoria
64	Wodonga	Victoria
65	Perth	Western Australia
66	Albany	Western Australia

	Location	State
67	Bunbury	Western Australia
68	Busselton	Western Australia
69	Fremantle	Western Australia
70	Geraldton	Western Australia
71	Joondalup	Western Australia
72	Kalgoorlie	Western Australia
73	Karratha	Western Australia
74	Mandurah	Western Australia
75	Rockingham	Western Australia

76 rows × 2 columns

In [50]:
```
state_data = data.merge(state, on = "Location", how = "left")
state_data
```

Out[50]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxte
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	2008	12	1	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	2008	12	2	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	2008	12	3	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	2008	12	4	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	2008	12	5	

In [ ]:

In [ ]:

In [ ]:

Example 6.1: pivot tables

Using pivot tables find the average monthly rainfall in the year 2016 of all the locations. The information can then be used to predict the sales of tea in the year 2017.

In [51]:
```
data_2016 = data[data["Year"] ==2016]
data_2016
```

Out[51]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	Year	Month	Dayofmonth	Maxte
2474	2016-01-01	Albury	20.4	37.6	0.0	NaN	NaN	ENE	54.0	2016	1	1	
2475	2016-01-02	Albury	20.9	33.6	0.4	NaN	NaN	SSE	50.0	2016	1	2	
2476	2016-01-03	Albury	18.4	23.1	2.2	NaN	NaN	ENE	48.0	2016	1	3	
2477	2016-01-04	Albury	17.3	23.7	15.6	NaN	NaN	SSE	39.0	2016	1	4	
2478	2016-01-05	Albury	15.5	22.9	6.8	NaN	NaN	ENE	31.0	2016	1	5	
2479	2016-01-06	Albury	17.0	28.1	0.2	NaN	NaN	SE	39.0	2016	1	6	
2480	2016-01-07	Albury	16.4	28.0	0.0	NaN	NaN	SE	35.0	2016	1	7	

In [52]:
```
data_2016.pivot_table(index = "Location", columns = "Month", values = "Rainfall", aggfunc='mean')
```

Out[52]:

Month Location	1	2	3	4	5	6	7	8	9	10	11	12
Adelaide	1.703226	0.634483	1.735484	0.320000	2.838710	3.173333	3.612903	1.896774	4.373333	2.612903	1.106667	2.800000
Albany	2.380645	0.748276	1.144828	3.153333	3.158065	4.010000	3.954839	3.777419	3.426667	2.025806	0.753333	0.650000
Albury	2.206452	1.013793	0.961290	0.546667	3.477419	2.866667	3.767742	2.400000	4.740000	1.980645	1.653333	0.735484
AliceSprings	1.290323	0.910345	0.522581	0.000000	1.832258	0.933333	0.000000	0.658065	1.640000	0.109677	0.233333	4.352000
BadgerysCreek	5.012903	0.441379	1.019355	0.346667	0.380645	8.346667	1.438710	1.890323	1.826667	0.458065	0.337931	0.728571
Ballarat	1.358621	0.355556	1.180645	0.460000	2.303226	2.353333	3.012903	2.051613	5.940000	3.303226	1.133333	0.890323
Bendigo	1.117241	0.162963	1.077419	0.493333	2.677419	1.913333	2.961290	2.612903	5.113333	1.883871	1.253333	0.941935
Brisbane	0.972414	0.523077	3.787097	0.426667	0.890323	8.826667	0.909677	1.019355	1.726667	0.974194	0.933333	3.290323
Cairns	9.316129	4.689655	8.329032	4.593103	9.336842	2.220000	2.741935	1.348387	4.500000	2.427273	0.765217	4.033333
Canberra	3.432258	0.806897	0.916129	0.226667	1.535484	4.806667	2.290323	1.490323	4.973333	1.406452	1.893333	2.083871
Cobar	0.728571	0.062069	2.344828	1.313333	2.438710	3.586667	1.058065	1.948387	3.406667	0.658065	1.126316	0.993548
CoffsHarbour	3.129032	1.086207	2.345161	3.876667	0.112903	11.350000	0.732258	5.706452	1.163333	0.841935	3.546667	2.051613
Dartmoor	0.606897	1.420690	0.922581	1.133333	3.238710	5.053333	4.116129	2.780645	3.640000	2.341935	1.560000	1.670968
Darwin	5.477419	5.186207	3.735484	1.546667	2.012903	0.000000	0.000000	0.000000	2.146667	4.219355	6.940000	13.122581
GoldCoast	3.360000	2.336000	1.745455	3.764286	0.600000	10.042857	0.607407	2.407407	1.284615	1.264000	0.992308	1.825806
Hobart	1.296774	0.751724	0.548387	0.293333	3.096774	3.846667	2.554839	0.509677	3.140000	2.690323	2.173333	1.825806
Katherine	8.112903	3.889655	3.245161	0.626667	1.438710	0.000000	0.179310	0.000000	1.933333	0.141935	1.350000	3.469565
Launceston	4.961290	0.848276	1.596774	0.920000	4.312903	4.483333	4.545161	2.083871	2.706667	2.377419	1.840000	2.287097
Melbourne	NaN	NaN	NaN	NaN	1.838710	1.813333	2.070968	1.974194	2.906667	2.045161	1.046667	1.277419
MelbourneAirport	1.445161	0.151724	0.877419	1.213333	1.516129	2.213333	2.045161	0.987097	3.326667	2.496774	1.326667	2.496774
Mildura	2.200000	0.006897	0.012903	0.100000	1.400000	0.606667	0.619355	0.993548	3.120000	0.800000	1.706667	0.180645
Moree	3.270588	0.110345	0.347826	0.453333	1.122581	2.106667	0.470968	1.612903	4.620000	1.832258	0.773333	1.135484
MountGambier	0.425806	1.668966	0.916129	1.046667	3.438710	5.433333	4.948387	2.779310	3.921429	2.290323	1.653333	1.917241
MountGinini	4.690323	1.427586	1.767742	0.520000	3.432258	9.153333	6.354839	2.277419	7.110345	3.600000	1.926667	2.967742
Newcastle	13.851613	2.475000	2.280645	2.140000	0.458065	6.713333	1.787097	1.674194	1.866667	1.813793	1.693333	1.768000

Month	1	2	3	4	5	6	7	8	9	10	11	12
Location												
Nhil	1.193548	0.496296	1.116129	0.220000	1.322581	1.446667	1.993548	1.083871	3.113333	1.903226	0.453333	1.032258
NorahHead	7.013793	0.600000	4.980645	1.813333	0.412903	6.506667	2.664516	2.432258	2.306667	1.941935	1.173333	2.238710
NorfolkIsland	4.419355	9.165517	8.841379	2.740000	3.880645	3.920000	2.161290	3.516129	2.440000	1.290323	7.084211	0.375000
Nuriootpa	0.967742	0.424138	1.470968	0.483333	2.862069	3.140000	2.437931	2.054839	6.036667	1.512903	0.783333	1.658065
PearceRAAF	0.696552	0.034483	3.283871	2.273333	2.909677	3.140000	3.845161	4.329032	1.664286	1.307143	0.246667	0.445161
Penrith	9.929032	0.072000	0.541935	0.406667	0.268966	6.980000	1.776000	1.574194	1.666667	0.400000	0.585714	2.248276
Perth	0.490323	0.020690	0.541935	2.273333	3.612903	3.646667	4.503226	3.974194	2.293333	1.206452	0.480000	0.329032
PerthAirport	0.754839	0.027586	0.690323	2.053333	3.419355	2.880000	4.174194	4.245161	2.060000	1.219355	0.186667	0.277419
Portland	0.944828	2.806897	0.961290	1.386667	4.658065	4.000000	5.425806	3.522581	5.471429	3.703226	2.246667	1.524138
Richmond	8.593548	0.525926	0.441379	0.286667	0.135484	6.306667	1.845161	1.987097	1.846667	0.438710	1.353333	2.529032
Sale	3.613793	0.213793	1.954839	0.866667	1.219355	2.433333	3.303226	0.845161	1.207143	2.259259	1.780000	0.761290
SalmonGums	1.331034	0.496552	1.670968	1.093333	1.148387	1.906667	1.051613	1.729032	0.900000	0.374194	0.113333	0.593103
Sydney	8.058065	0.889655	6.232258	5.166667	0.232258	10.166667	3.374194	4.883871	2.333333	1.012903	0.906667	2.096774
SydneyAirport	8.477419	1.393103	5.025806	2.700000	0.432258	9.313333	3.683871	4.238710	2.260000	1.103226	0.893333	1.883871
Townsville	2.470968	3.262069	18.000000	0.700000	0.012903	1.966667	1.716129	0.677419	0.242857	0.264516	0.366667	1.064286
Tuggeranong	3.234483	1.162963	1.875862	0.235714	1.200000	6.133333	2.077419	1.232258	5.033333	1.392593	1.780000	3.012903
Uluru	0.081481	0.491667	0.935484	0.006667	2.406452	2.681481	0.690323	0.935484	0.520000	0.064516	0.046667	6.929032
WaggaWagga	1.780645	0.558621	1.077419	0.360000	3.813793	2.820000	2.987097	1.896774	5.700000	2.064516	1.041667	0.600000
Walpole	3.584000	1.427586	1.151724	4.221429	4.870968	5.408333	6.379310	6.448276	5.092857	2.928571	0.938462	1.703226
Watsonia	1.207407	0.324138	0.890323	1.700000	1.722581	2.393333	2.051613	1.780645	2.792857	3.048276	2.033333	3.800000
Williamtown	13.625806	1.117241	1.474074	5.585185	0.361290	5.230000	1.696774	1.800000	1.660000	2.406452	1.561538	11.680000
Witchcliffe	0.296296	0.324138	0.551724	1.735714	6.238710	7.650000	7.806452	7.348387	4.364286	2.751724	0.570370	0.832258
Wollongong	5.735484	1.710345	6.587097	1.820000	0.316129	14.553333	3.483871	3.612903	1.580000	0.200000	0.986207	1.586667
Woomera	0.012903	0.524138	0.774194	0.033333	0.945161	1.890000	0.222581	0.890323	1.500000	0.335484	0.016667	0.505000

Find the Pandas pivot table documentation [here (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot_table.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.pivot_table.html)

This information can be used to decide the stocks of tea in each of the stores.

You can modify the pivot_table command to get a lot of work done quickly.

In [53]:
```
data_2016.pivot_table(index = "Location", columns = "Month", values = "Sunshine", aggfunc='mean')
```

Out[53]:

Month	1	2	3	4	5	6	7	8	9	10	11	1
Location												
Albany	5.588000	7.825000	4.450000	3.473333	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
AliceSprings	9.427273	11.638462	9.558621	10.212500	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
Brisbane	8.870000	9.781481	8.558065	7.853333	8.332258	5.400000	6.864516	8.577419	7.690000	9.909677	10.293103	8.80645
Cairns	8.576667	9.020690	7.116667	6.980952	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
Dartmoor	8.873333	7.813793	5.293548	5.304545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
Darwin	8.045161	8.065517	7.490323	9.320000	9.416129	10.266667	10.329032	10.383871	8.633333	9.145161	8.433333	5.92258
Hobart	6.900000	8.186207	6.335484	6.420000	4.287097	3.630000	4.993548	6.574194	6.043333	7.587097	7.413333	8.43000
Melbourne	NaN	NaN	NaN	8.400000	5.219355	3.850000	3.861290	5.738710	4.893333	6.941935	6.960000	8.37419
MelbourneAirport	6.912903	8.472414	5.835484	6.373333	5.219355	3.850000	3.861290	5.738710	4.893333	6.941935	6.960000	8.37419
Mildura	8.990000	11.496429	9.067742	7.892857	6.057143	4.950000	4.966667	7.377419	7.000000	9.270000	9.709524	10.24583
Moree	9.623077	11.200000	10.270000	9.272727	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
MountGambier	9.000000	7.813793	5.293548	5.304545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
NorfolkIsland	7.643333	4.710714	6.546429	6.538462	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
Nuriootpa	10.167742	10.537931	7.593548	7.306667	4.831579	4.810000	4.393103	6.722222	5.353571	8.970968	10.072414	10.06071
PearceRAAF	9.330000	11.641379	9.003226	6.413333	6.590323	5.686667	6.141935	6.193548	7.862069	9.762069	11.940000	12.04516
Perth	9.412903	11.641379	9.003226	6.413333	6.590323	5.686667	6.141935	6.193548	7.940000	9.725806	11.940000	12.04516
PerthAirport	9.412903	11.641379	9.003226	6.413333	6.590323	5.686667	6.141935	6.193548	7.940000	9.725806	11.940000	12.04516
Portland	8.900000	7.813793	5.293548	5.304545	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
Sydney	6.600000	9.572414	7.412903	7.246667	7.451613	5.631034	6.663333	6.830000	7.466667	9.106452	9.500000	7.42580
SydneyAirport	6.600000	9.572414	7.412903	7.246667	7.451613	5.631034	6.663333	6.830000	7.466667	9.106452	9.500000	7.42580
Townsville	9.480645	9.210345	6.912903	8.600000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
WaggaWagga	8.151613	11.500000	9.145161	8.957692	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
Watsonia	6.924138	8.472414	5.835484	6.373333	5.219355	3.850000	3.861290	5.738710	4.879310	7.010000	6.960000	8.10000
Williamtown	7.226667	10.913333	8.181250	6.723077	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Month	1	2	3	4	5	6	7	8	9	10	11	1:
Location												
Woomera	10.387097	11.710345	7.609677	8.957692	7.644828	NaN	NaN	NaN	NaN	NaN	0.000000	Nal

Note

[Here (https://pandas.pydata.org/pandas-docs/stable/index.html)](https://pandas.pydata.org/pandas-docs/stable/index.html) is the link to the official documentation of Pandas. Be sure to visit it inorder to explore to availability of functions in the library.

In [ ]:

```

```

Create DataFrames

Since a new concept is being introduced, it is beneficial to explore the concept first using simple DataFrames. Once you understand the usage and the capabilities of these concepts, you can think of ways to apply these capabilities as and when needed.

In [1]:
```
import pandas as pd
```

In [2]:
```
df_1 = {"col1":[1,2,3,4], "col2": [5,6,7,8]}
df_2 = {"col1":[11,12,13,14], "col2": [15,16,17,18]}
```

In [3]:
```
df1 = pd.DataFrame(df_1)
df2 = pd.DataFrame(df_2)
```

In [4]:
```
df1
```

Out[4]:

	col1	col2
0	1	5
1	2	6
2	3	7
3	4	8

In [5]:
```
df2
```

Out[5]:

	col1	col2
0	11	15
1	12	16
2	13	17
3	14	18

Concatenation

It is used when you want to stick two dataframes together without any consideration given to matching elements. In contrast, the merge command uses a key to stitch two data frames together.

If the shape of the two concatenating dataframes does not match, NaN values are added to make the dimensions uniform.

In [7]:
```
pd.concat([df1, df2], axis = 0)

# Axis 0 represents row wise concatenation
```

Out[7]:

	col1	col2
0	1	5
1	2	6
2	3	7
3	4	8
0	11	15
1	12	16
2	13	17
3	14	18

NOTE

- Rows in df2 get added to the df1
- Intexes of df2 remain the same as they were before the join.

In [8]:
```
pd.concat([df1, df2], axis = 1)

# Axis 0 represents column wise concatenation
```

Out[8]:

	col1	col2	col1	col2
0	1	5	11	15
1	2	6	12	16
2	3	7	13	17
3	4	8	14	18

In [9]:
```
df1["col3"] = df1["col1"] + df1["col2"]

# After this operation df1 will have 3 columns while df2 has only 2.
```

In [10]:
```
pd.concat([df1, df2], axis = 0)
```

Out[10]:

	col1	col2	col3
0	1	5	6.0
1	2	6	8.0
2	3	7	10.0
3	4	8	12.0
0	11	15	NaN
1	12	16	NaN
2	13	17	NaN
3	14	18	NaN

Since there is one extra column in df1, the corresponding vales in df2 become `NaN` or null values.

Arithmetic Operators on DataFrames

You can perform element wise operations on dataframes as well. These are very similar to operations you performed on NumPy arrays.

for example, if you want to add all the elements on `df1` to the correspopnding elements on `df2` you can use the '+' operator.

In [11]:
```
df1 + df2
```

Out[11]:

	col1	col2	col3
0	12	20	NaN
1	14	22	NaN
2	16	24	NaN
3	18	26	NaN

As you saw all the elements in `df1` got added to corresponding elements in `df2`

But the `df1` had three columns while `df2` had two. So the operation for the third column is incomplete, that is why you see the null values in the result. This is the most significant difference in using operators in pandas and NumPy; this operation would have thrown an error if it was executed using NumPy arrays.

The same result can be achieved by the `add()` method

In [13]:
```
df1.add(df2)
```

Out[13]:

	col1	col2	col3
0	12	20	NaN
1	14	22	NaN
2	16	24	NaN
3	18	26	NaN

Along with the normal addition this add method also provides additional functionalities. You can read about them [here (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.add.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.add.html)

similar to the '+' operator and the `add()` there are other operators as well

- `sub()` : '-'
- `mul()` : '*'
- `div()` : '/'
- `floordiv()` : '//'
- `mod()` : '%'
- `pow()` : '**'

In [15]:

```
# recreating the DataFrames so that the dimentions match.

df_1 = {"col1":[1,2,3,4], "col2": [5,6,7,8]}
df_2 = {"col1":[11,12,13,14], "col2": [15,16,17,18]}

df1 = pd.DataFrame(df_1)
df2 = pd.DataFrame(df_2)

print (df1)
print (df2)
```

```
   col1  col2
0     1     5
1     2     6
2     3     7
3     4     8
   col1  col2
0    11    15
1    12    16
2    13    17
3    14    18
```

In [16]:
```
df2 - df1
```

Out[16]:

	col1	col2
0	10	10
1	10	10
2	10	10
3	10	10

In [17]:
```
df2 ** df1
```

Out[17]:

	col1	col2
0	11	759375
1	144	16777216
2	2197	410338673
3	38416	11019960576

In [24]:
```
# recreating the DataFrames so that the dimentions match.

df_1 = {"col1":[1,2,3,4], "col2": [5,6,7,8]}
df_2 = {"col1":[11,12,13,14]}

df1 = pd.DataFrame(df_1)
df2 = pd.DataFrame(df_2)

print (df1)
print (df2)
```

```
   col1  col2
0     1     5
1     2     6
2     3     7
3     4     8
   col1
0    11
1    12
2    13
3    14
```

In [25]:
```
df1 + df2
```

Out[25]:

	col1	col2
0	12	NaN
1	14	NaN
2	16	NaN
3	18	NaN

One of the advantages of pandas DataFrame is that it can hold data of different data types.

Which leads us to the question What would happen of operators were used on DataFrames which have "non-numerical" data types?

In [19]:
```
df_1 = {"col1":[1,2,3,4], "col2": [5,6,7,8], "col3": [True,False,False,True], "col4": ["a","b","c","d"] }
df_2 = {"col1":[11,12,13,14], "col2": [15,16,17,18], "col3": [True,False,True,False], "col4": ["e","f","g","h"]}

df1 = pd.DataFrame(df_1)
df2 = pd.DataFrame(df_2)

print (df1)
print (df2)
```

```
   col1  col2   col3 col4
0     1     5   True    a
1     2     6  False    b
2     3     7  False    c
3     4     8   True    d
   col1  col2   col3 col4
0    11    15   True    e
1    12    16  False    f
2    13    17   True    g
3    14    18  False    h
```

In [20]:
```
df1 +df2
```

```
D:\Software\Anaconda\lib\site-packages\pandas\core\computation\expressions.py:178: UserWarning: evaluating in Python
space because the '+' operator is not supported by numexpr for the bool dtype, use '|' instead
  f"evaluating in Python space because the {repr(op_str)} "
```

Out[20]:

	col1	col2	col3	col4
0	12	20	True	ae
1	14	22	False	bf
2	16	24	True	cg
3	18	26	True	dh

Something very interesting has happened.

Pandas was smart enough to recognise the different data types and use the operators accordingly.

- For int data type, it performed addition

- For boolean, it performed OR operation
- For string, it performed concatenation

In [21]:
```
df1 - df2
```

```
D:\Software\Anaconda\lib\site-packages\pandas\core\computation\expressions.py:178: UserWarning: evaluating in Pyth
on space because the '-' operator is not supported by numexpr for the bool dtype, use '^' instead
  f"evaluating in Python space because the {repr(op_str)} "

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
D:\Software\Anaconda\lib\site-packages\pandas\core\ops\array_ops.py in na_arithmetic_op(left, right, op, str_rep)
    148     try:
--> 149         result = expressions.evaluate(op, str_rep, left, right)
    150     except TypeError:

D:\Software\Anaconda\lib\site-packages\pandas\core\computation\expressions.py in evaluate(op, op_str, a, b, use_nu
mexpr)
    208         return _evaluate(op, op_str, a, b)
--> 209     return _evaluate_standard(op, op_str, a, b)
    210

D:\Software\Anaconda\lib\site-packages\pandas\core\computation\expressions.py in _evaluate_standard(op, op_str, a,
b)
     69     with np.errstate(all="ignore"):
```

This throws an error because there is not '-' in strings and pandas cannot figure out what to do.

In [22]:
```
df_1 = {"col1":[1,2,3,4], "col2": [5,6,7,8], "col3": [True,False,False,True], "col4": ["a","b","c","d"] }
df_2 = {"col1": [True,False,True,False], "col2": ["e","f","g","h"], "col3":[11,12,13,14], "col4": [15,16,17,18] }

df1 = pd.DataFrame(df_1)
df2 = pd.DataFrame(df_2)

print (df1)
print (df2)
```

```
   col1  col2   col3 col4
0     1     5   True    a
1     2     6  False    b
2     3     7  False    c
3     4     8   True    d
    col1 col2  col3  col4
0   True    e    11    15
1  False    f    12    16
2   True    g    13    17
3  False    h    14    18
```

In [23]:
```
df1 + df2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
D:\Software\Anaconda\lib\site-packages\pandas\core\ops\array_ops.py in na_arithmetic_op(left, right, op, str_rep)
    148     try:
--> 149         result = expressions.evaluate(op, str_rep, left, right)
    150     except TypeError:

D:\Software\Anaconda\lib\site-packages\pandas\core\computation\expressions.py in evaluate(op, op_str, a, b, use_nu
mexpr)
    207     if use_numexpr:
--> 208         return _evaluate(op, op_str, a, b)
    209     return _evaluate_standard(op, op_str, a, b)

D:\Software\Anaconda\lib\site-packages\pandas\core\computation\expressions.py in _evaluate_numexpr(op, op_str, a,
b)
    120     if result is None:
--> 121         result = _evaluate_standard(op, op_str, a, b)
    122
```

Since the data types of correcponding columns do not match Pandas throws a type error.

Summary

1. ***`Concatenation`*** ***: Used when you want to stich to dataframes together without any reguard to the values.***

a. Even if the shapes do not match the operation is performed. Filling Null values wherever necessary.

2. ***`operators`*** ***: Can perform element wise operations on Pandas DataFrames.***

a. You can use operators themselves '+' or the function `add()` for the same result.
b. If the Shape does not match then null values are added. c. Can work with differnet data types as well, as long as the operation is defined for that data type.