Pandas Datastructures

Series

- Creating series
- Accessing data in series
- Adding data in a series
- Deleting data from a series
- Changing the dtypes in a series

In [1]:

```
l1 = range(0,10)
print(list(l1))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [2]:

```
# Adding 1 to each element of l1

l1+1

# doesn't support vectoriation
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-e89020f189dc> in <module>
      1 # Adding 1 to each element of l1
      2
----> 3 l1+1
      4
      5 # doesn't support vectoriation

TypeError: unsupported operand type(s) for +: 'range' and 'int'
```

In [3]:

```
# 1 : for loop

l2 = []

for x in l1:
    l2.append(x+1)

print(l2)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [4]:

```
# 2 : list comprehension

[ x+1 for x in l1 ]
```

Out[4]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [5]:

```
# 3 = map-lambda function

l3 = map(lambda x : x+1 , l1 )

print(list(l3))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Creating a Series object

- General syntax is :
- s = Series(data,index)
- data : Dictionary, array, scalar

In [6]:

```
import pandas as pd
from pandas import Series , DataFrame
import numpy as np
```

Series from an Array / List

In [7]:

```
s = Series([1,2,3,4,5])
s
```

Out[7]:

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

In [8]:

```
print(s[0])
print(s[2])
```

```
1
3
```

In [9]:

```
print(s.values)
print(s.index)
```

```
[1 2 3 4 5]
RangeIndex(start=0, stop=5, step=1)
```

In [10]:

```
print(list(s.index))
```

```
[0, 1, 2, 3, 4]
```

In [11]:

```
print(dir(s))
```

```
['T', '_AXIS_ALIASES', '_AXIS_IALIASES', '_AXIS_LEN', '_AXIS_NAMES', '_AXIS_
NUMBERS', '_AXIS_ORDERS', '_AXIS_REVERSED', '_HANDLED_TYPES', '__abs__', '__
add__', '__and__', '__annotations__', '__array__', '__array_priority__', '__
array_ufunc__', '__array_wrap__', '__bool__', '__class__', '__contains__',
'__copy__', '__deepcopy__', '__delattr__', '__delitem__', '__dict__', '__dir
__', '__div__', '__divmod__', '__doc__', '__eq__', '__finalize__', '__float_
_', '__floordiv__', '__format__', '__ge__', '__getattr__', '__getattribute_
_', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__', '__iand
__', '__ifloordiv__', '__imod__', '__imul__', '__init__', '__init_subclass_
_', '__int__', '__invert__', '__ior__', '__ipow__', '__isub__', '__iter__',
'__itruediv__', '__ixor__', '__le__', '__len__', '__long__', '__lt__', '__ma
tmul__', '__mod__', '__module__', '__mul__', '__ne__', '__neg__', '__new__',
'__nonzero__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rd
iv__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordi
v__', '__rmatmul__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow
__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__setitem__',
'__setstate__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__t
ruediv__', '__weakref__', '__xor__', '_accessors', '_add_numeric_operation
s', '_add_series_or_dataframe_operations', '_agg_by_level', '_agg_examples_d
oc', '_agg_see_also_doc', '_aggregate', '_aggregate_multiple_funcs', '_align
_frame', '_align_series', '_binop', '_box_item_values', '_builtin_table', '_
can_hold_na', '_check_inplace_setting', '_check_is_chained_assignment_possib
le', '_check_label_or_level_ambiguity', '_check_setitem_copy', '_clear_item_
cache', '_clip_with_one_bound', '_clip_with_scalar', '_consolidate', '_conso
lidate_inplace', '_construct_axes_dict', '_construct_axes_dict_from', '_cons
truct_axes_from_arguments', '_constructor', '_constructor_expanddim', '_cons
tructor_sliced', '_convert', '_convert_dtypes', '_create_indexer', '_cython_
table', '_deprecations', '_dir_additions', '_dir_deletions', '_drop_axis',
'_drop_labels_or_levels', '_ensure_type', '_find_valid_index', '_from_axes',
'_get_axis', '_get_axis_name', '_get_axis_number', '_get_axis_resolvers', '_
get_block_manager_axis', '_get_bool_data', '_get_cacher', '_get_cleaned_colu
mn_resolvers', '_get_cython_func', '_get_index_resolvers', '_get_item_cach
e', '_get_label_or_level_values', '_get_numeric_data', '_get_value', '_get_v
alues', '_get_values_tuple', '_get_with', '_gotitem', '_iget_item_cache', '_
index', '_indexed_same', '_info_axis', '_info_axis_name', '_info_axis_numbe
r', '_init_dict', '_init_mgr', '_internal_get_values', '_internal_names', '_
internal_names_set', '_is_builtin_func', '_is_cached', '_is_copy', '_is_date
like_mixed_type', '_is_label_or_level_reference', '_is_label_reference', '_i
s_level_reference', '_is_mixed_type', '_is_numeric_mixed_type', '_is_view',
'_ix', '_ixs', '_map_values', '_maybe_cache_changed', '_maybe_update_cache
r', '_metadata', '_ndarray_values', '_needs_reindex_multi', '_obj_with_exclu
sions', '_protect_consolidate', '_reduce', '_reindex_axes', '_reindex_indexe
r', '_reindex_multi', '_reindex_with_indexers', '_repr_data_resource_', '_re
pr_latex_', '_reset_cache', '_reset_cacher', '_selected_obj', '_selection',
'_selection_list', '_selection_name', '_set_as_cached', '_set_axis', '_set_a
xis_name', '_set_is_copy', '_set_item', '_set_labels', '_set_name', '_set_su
btyp', '_set_value', '_set_values', '_set_with', '_set_with_engine', '_setup
_axes', '_slice', '_stat_axis', '_stat_axis_name', '_stat_axis_number', '_ta
ke_with_is_copy', '_to_dict_of_blocks', '_try_aggregate_string_function', '_
typ', '_unpickle_series_compat', '_update_inplace', '_validate_dtype', '_val
ues', '_where', '_xs', 'abs', 'add', 'add_prefix', 'add_suffix', 'agg', 'agg
regate', 'align', 'all', 'any', 'append', 'apply', 'argmax', 'argmin', 'args
ort', 'array', 'asfreq', 'asof', 'astype', 'at', 'at_time', 'attrs', 'autoco
rr', 'axes', 'between', 'between_time', 'bfill', 'bool', 'clip', 'combine',
'combine_first', 'convert_dtypes', 'copy', 'corr', 'count', 'cov', 'cummax',
'cummin', 'cumprod', 'cumsum', 'describe', 'diff', 'div', 'divide', 'divmo
d', 'dot', 'drop', 'drop_duplicates', 'droplevel', 'dropna', 'dtype', 'dtype
```

```
s', 'duplicated', 'empty', 'eq', 'equals', 'ewm', 'expanding', 'explode', 'f
actorize', 'ffill', 'fillna', 'filter', 'first', 'first_valid_index', 'floor
div', 'ge', 'get', 'groupby', 'gt', 'hasnans', 'head', 'hist', 'iat', 'idxma
x', 'idxmin', 'iloc', 'index', 'infer_objects', 'interpolate', 'is_monotoni
c', 'is_monotonic_decreasing', 'is_monotonic_increasing', 'is_unique', 'isi
n', 'isna', 'isnull', 'item', 'items', 'iteritems', 'keys', 'kurt', 'kurtosi
s', 'last', 'last_valid_index', 'le', 'loc', 'lt', 'mad', 'map', 'mask', 'ma
x', 'mean', 'median', 'memory_usage', 'min', 'mod', 'mode', 'mul', 'multipl
y', 'name', 'nbytes', 'ndim', 'ne', 'nlargest', 'notna', 'notnull', 'nsmalle
st', 'nunique', 'pct_change', 'pipe', 'plot', 'pop', 'pow', 'prod', 'produc
t', 'quantile', 'radd', 'rank', 'ravel', 'rdiv', 'rdivmod', 'reindex', 'rein
dex_like', 'rename', 'rename_axis', 'reorder_levels', 'repeat', 'replace',
'resample', 'reset_index', 'rfloordiv', 'rmod', 'rmul', 'rolling', 'round',
'rpow', 'rsub', 'rtruediv', 'sample', 'searchsorted', 'sem', 'set_axis', 'sh
ape', 'shift', 'size', 'skew', 'slice_shift', 'sort_index', 'sort_values',
'squeeze', 'std', 'sub', 'subtract', 'sum', 'swapaxes', 'swaplevel', 'tail',
'take', 'to_clipboard', 'to_csv', 'to_dict', 'to_excel', 'to_frame', 'to_hd
f', 'to_json', 'to_latex', 'to_list', 'to_markdown', 'to_numpy', 'to_perio
d', 'to_pickle', 'to_sql', 'to_string', 'to_timestamp', 'to_xarray', 'transf
orm', 'transpose', 'truediv', 'truncate', 'tshift', 'tz_convert', 'tz_locali
ze', 'unique', 'unstack', 'update', 'value_counts', 'values', 'var', 'view',
'where', 'xs']
```

In [12]:

```
# Syntax : s1 = Series(data,index)

s1 = Series([1,2,3,4,5,] , ['a','b','c','d','e'])
s1
```

Out[12]:

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

In [13]:

```
print(s1[0])
print(s1['a'])
```

```
1
1
```

In [14]:

```
s1.index
```

Out[14]:

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [15]:

```
# Adding 1 element to the series

s1['f'] = 6
s1
```

Out[15]:

```
a    1
b    2
c    3
d    4
e    5
f    6
dtype: int64
```

In [16]:

```
# Adding multiple elements to the series
# We will use append method to append sub series to bigger series

ss1 = Series( [7,8] , ['g','h'] )
s1 = s1.append(ss1)
s1
```

Out[16]:

```
a    1
b    2
c    3
d    4
e    5
f    6
g    7
h    8
dtype: int64
```

In [17]:

```
# Deleting 1 element from the series

s1 = s1.drop('f')
s1
```

Out[17]:

```
a    1
b    2
c    3
d    4
e    5
g    7
h    8
dtype: int64
```

In [18]:

```
# Deleting multiple elements from the series

s1 = s1.drop(['g','h'])
s1
```

Out[18]:

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Series from a Dictionary

In [19]:

```
d1 = {'a': 2 , 'b': 3}
s3 = Series(d1)
s3
```

Out[19]:

```
a    2
b    3
dtype: int64
```

In [20]:

```
# We can make in-place changes to the series

s3['a'] = 4
s3[1] = 5

s3
```

Out[20]:

```
a    4
b    5
dtype: int64
```

In [21]:

```
# Series supports vectorization

s3+1
```

Out[21]:

```
a    5
b    6
dtype: int64
```

In [22]:

```
s3*2
```

Out[22]:

```
a     8
b    10
dtype: int64
```

In [23]:

```
s3**2
```

Out[23]:

```
a    16
b    25
dtype: int64
```

In [24]:

```
np.log(s3)
```

Out[24]:

```
a    1.386294
b    1.609438
dtype: float64
```

In [25]:

```
s1+s3
```

Out[25]:

```
a    5.0
b    7.0
c    NaN
d    NaN
e    NaN
dtype: float64
```

In [26]:

```
#Series is built upon numpy arrays, one can efficiently change the dtype of the numpy array

s4 = Series(['1','2','3','4','5','6'])
s4
```

Out[26]:

```
0    1
1    2
2    3
3    4
4    5
5    6
dtype: object
```

In [27]:

```
s4.dtype
```

Out[27]:

```
dtype('O')
```

In [28]:

```
s4 = s4.astype('float')
s4.dtype
```

Out[28]:

```
dtype('float64')
```

In [29]:

```
s4
```

Out[29]:

```
0    1.0
1    2.0
2    3.0
3    4.0
4    5.0
5    6.0
dtype: float64
```

In [30]:

```
##Corner case

s5 = Series(['1',2,3,4,np.nan,np.nan])
s5
```

Out[30]:

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

In [31]:

```
s5 = s5.astype('int64')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-31-569c857e3218> in <module>
----> 1 s5 = s5.astype('int64')

~\Downloads\Swapnil\Data Analytics\anaconda3\lib\site-packages\pandas\core\g
eneric.py in astype(self, dtype, copy, errors)
   5696         else:
   5697             # else, only a single dtype is given
-> 5698             new_data = self._data.astype(dtype=dtype, copy=copy, err
ors=errors)
   5699             return self._constructor(new_data).__finalize__(self)
   5700

~\Downloads\Swapnil\Data Analytics\anaconda3\lib\site-packages\pandas\core\i
nternals\managers.py in astype(self, dtype, copy, errors)
    580
    581     def astype(self, dtype, copy: bool = False, errors: str = "rais
e"):
--> 582         return self.apply("astype", dtype=dtype, copy=copy, errors=e
rrors)
    583
    584     def convert(self, **kwargs):

~\Downloads\Swapnil\Data Analytics\anaconda3\lib\site-packages\pandas\core\i
nternals\managers.py in apply(self, f, filter, **kwargs)
    440                 applied = b.apply(f, **kwargs)
    441             else:
--> 442                 applied = getattr(b, f)(**kwargs)
    443             result_blocks = _extend_blocks(applied, result_blocks)
    444

~\Downloads\Swapnil\Data Analytics\anaconda3\lib\site-packages\pandas\core\i
nternals\blocks.py in astype(self, dtype, copy, errors)
    623             vals1d = values.ravel()
    624             try:
--> 625                 values = astype_nansafe(vals1d, dtype, copy=True)
    626             except (ValueError, TypeError):
    627                 # e.g. astype_nansafe can fail on object-dtype of st
rings

~\Downloads\Swapnil\Data Analytics\anaconda3\lib\site-packages\pandas\core\d
types\cast.py in astype_nansafe(arr, dtype, copy, skipna)
    872         # work around NumPy brokenness, #1987
    873         if np.issubdtype(dtype.type, np.integer):
--> 874             return lib.astype_intsafe(arr.ravel(), dtype).reshape(ar
r.shape)
    875
    876         # if we have a datetime/timedelta array of objects

pandas\_libs\lib.pyx in pandas._libs.lib.astype_intsafe()

ValueError: cannot convert float NaN to integer
```

In [32]:

```
s5 = s5.astype('float64')
s5
```

Out[32]:

```
0    1.0
1    2.0
2    3.0
3    4.0
4    NaN
5    NaN
dtype: float64
```

In [33]:

```
type(s5)
```

Out[33]:

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

DataFrames

DataFrames are the workhorse of pandas and are directly inspired by the R programming language. We can think of a DataFrame as a bunch of Series objects put together to share the same index.

- Creating data frames
- Accessing rows and columns of a data frame
- Changing name of a dataframe column
- Basic DataFrame methods
- Changing dtypes of a dataframe column
- When analysing data apart from the ability to work with vectorized data, the ability to work with the tabular data is also required. DataFrames is the tabular datastructure used to work with data.

Creating a dataframe

- From a dictionary of lists, dictionaries, series
- From a list of dictionaries

In [34]:

```
# From Dictionary of Lists

d1 = { 'sp' : [100,200,300,400] , 'profit' : [10,20,30,40] }

df1 = DataFrame(d1)
print(df1)
```

```
    sp  profit
0  100      10
1  200      20
2  300      30
3  400      40
```

In [35]:

```
# From Dictionary of Dictionaries

d2 = { 'sp' : { 'r1': 100 , 'r2': 200} , 'profit':{ 'r1': 10 , 'r2': 20 } }
df2 = DataFrame(d2)
print(df2)
```

```
     sp  profit
r1  100      10
r2  200      20
```

In [36]:

```
# From Dictionary of Series

d3 = { 'sp' : Series([100,200,300,400]) , 'profit' : Series([10,20,30,40]) }

df3 = DataFrame(d3)
print(df3)
```

```
    sp  profit
0  100      10
1  200      20
2  300      30
3  400      40
```

In [43]:

```
# From list of Dictionaries

l4 =  [{ 'sp': 100, 'profit': 10 , 'sales': 1000} , { 'sp': 200, 'profit': 20 , 'sales': 20

df4 = DataFrame(l4)
print(df4)
```

```
    sp  profit  sales
0  100      10   1000
1  200      20   2000
```

In [37]:

```
# Accessing rows and columns of DataFrame

print(df3['sp'])
type(df3['sp'])
```

```
0    100
1    200
2    300
3    400
Name: sp, dtype: int64
```

Out[37]:

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

In [38]:

```
print(df3[['sp','profit']])
type(df3[['sp','profit']])
```

```
    sp  profit
0  100      10
1  200      20
2  300      30
3  400      40
```

Out[38]:

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

In [39]:

```
# Renaming of the columns

print(df1)
```

```
    sp  profit
0  100      10
1  200      20
2  300      30
3  400      40
```

In [40]:

```
# We can use rename() method to change the column names , rename accepts a Dictionary

df1.rename(columns = { 'sp': 'Selling_price' , 'profit': 'Total_profit'})

df1
```

Out[40]:

	sp	profit
0	100	10
1	200	20
2	300	30
3	400	40

In [41]:

```
# corerct method

df1.rename(columns = { 'sp': 'Selling_price' , 'profit': 'Total_profit'} , inplace = True )
df1
```

Out[41]:

	Selling_price	Total_profit
0	100	10
1	200	20
2	300	30
3	400	40

Basic DataFrame Methods

In [42]:

```
df1.columns

# gives names of all the columns
```

Out[42]:

```
Index(['Selling_price', 'Total_profit'], dtype='object')
```

In [43]:

```
df1.head()

# gives first 5 rows of the dataframe
```

Out[43]:

	Selling_price	Total_profit
0	100	10
1	200	20
2	300	30
3	400	40

In [44]:

```
df1.tail()

# gives last 5 rows of the dataframe
```

Out[44]:

	Selling_price	Total_profit
0	100	10
1	200	20
2	300	30
3	400	40

In [45]:

```
df1.describe()

# gives out the summary
```

Out[45]:

	Selling_price	Total_profit
count	4.000000	4.000000
mean	250.000000	25.000000
std	129.099445	12.909944
min	100.000000	10.000000
25%	175.000000	17.500000
50%	250.000000	25.000000
75%	325.000000	32.500000
max	400.000000	40.000000

In [46]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Selling_price  4 non-null      int64
 1   Total_profit   4 non-null      int64
dtypes: int64(2)
memory usage: 192.0 bytes
```

In [47]:

```
df1.dtypes
```

Out[47]:

```
Selling_price    int64
Total_profit     int64
dtype: object
```

Changing dtypes of a dataframe

- Each column in a dataframe is a series object, we can extract the series object

In [48]:

```
df3
```

Out[48]:

	sp	profit
0	100	10
1	200	20
2	300	30
3	400	40

In [49]:

```
df3['sp'] = Series(df3['sp'].values.astype(float))
```

In [50]:

```
df3.dtypes
```

Out[50]:

```
sp        float64
profit      int64
dtype: object
```

In [51]:

```
df3
```

Out[51]:

	sp	profit
0	100.0	10
1	200.0	20
2	300.0	30
3	400.0	40

In [52]:

```
# A less verbose implementation

df3['profit'] = df3['profit'].astype(float)
```

In [53]:

```
df3.dtypes
```

Out[53]:

```
sp        float64
profit    float64
dtype: object
```

In [54]:

```
df3
```

Out[54]:

	sp	profit
0	100.0	10.0
1	200.0	20.0
2	300.0	30.0
3	400.0	40.0

In [55]:

```
df5 = DataFrame({'col1':['1','2','3','4','5'] , 'col2':['6','7','8','9','10']})
df5
```

Out[55]:

	col1	col2
0	1	6
1	2	7
2	3	8
3	4	9
4	5	10

In [56]:

```
df5.dtypes
```

Out[56]:

```
col1    object
col2    object
dtype: object
```

In [57]:

```
df5['col2'] = df5['col2'].astype(np.float)
df5.dtypes
```

Out[57]:

```
col1     object
col2    float64
dtype: object
```

In [58]:

```
df5
```

Out[58]:

	col1	col2
0	1	6.0
1	2	7.0
2	3	8.0
3	4	9.0
4	5	10.0

loc and iloc

- loc is label based : you have to specify the row and columns based on their row and column labels
- iloc is integer index based : you have to specify rows and columns based on their integer index

In [59]:

```
df3
```

Out[59]:

	sp	profit
0	100.0	10.0
1	200.0	20.0
2	300.0	30.0
3	400.0	40.0

In [60]:

```
df3.loc[3]['sp']
```

Out[60]:

```
400.0
```

In [61]:

```
df3.loc[[3,0]]['sp']
```

Out[61]:

```
3    400.0
0    100.0
Name: sp, dtype: float64
```

In [62]:

```
df3.iloc[3]['sp']
```

Out[62]:

```
400.0
```

In [63]:

```
df3.iloc[[3,0]]['sp']
```

Out[63]:

```
3    400.0
0    100.0
Name: sp, dtype: float64
```

In [64]:

```
df6 = DataFrame({'sp':[100,200,300,400,500] , 'profit':[10,20,30,40,50]} , index = list('ab
df6
```

Out[64]:

	sp	profit
a	100	10
b	200	20
c	300	30
d	400	40
e	500	50

In [65]:

```
df6.loc['e']
```

Out[65]:

```
sp        500
profit     50
Name: e, dtype: int64
```

In [66]:

```
df6.iloc[4]
```

Out[66]:

```
sp        500
profit     50
Name: e, dtype: int64
```

In [67]:

```
df6.loc[['e','a']]
```

Out[67]:

	sp	profit
e	500	50
a	100	10

In [68]:

```
df6.iloc[[4,1]]
```

Out[68]:

	sp	profit
e	500	50
b	200	20

In [69]:

```
df6.iloc[4,1]
```

Out[69]:

```
50
```

In [70]:

```
#loc and iloc also allow you to select both rows and columns from a DataFrame.

df6.loc[['e','b'],'profit']
```

Out[70]:

```
e    50
b    20
Name: profit, dtype: int64
```

In [71]:

```
df6.iloc[[4,1]]['profit']
```

Out[71]:

```
e    50
b    20
Name: profit, dtype: int64
```

In [72]:

```
print(df6.iloc[:,1])
type(df6.iloc[:,1])

# prints as series
```

```
a    10
b    20
c    30
d    40
e    50
Name: profit, dtype: int64
```

Out[72]:

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

In [73]:

```
print(df6.iloc[1,:])
type(df6.iloc[1,:])

# prints as series
```

```
sp        200
profit     20
Name: b, dtype: int64
```

Out[73]:

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

In [74]:

```
print(df6.loc[:,'profit'])
type(df6.loc[:,'profit'])

# prints as series
```

```
a    10
b    20
c    30
d    40
e    50
Name: profit, dtype: int64
```

Out[74]:

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

In [75]:

```
print(df6.iloc[:,[1]])
type(df6.iloc[:,[1]])

# prints as dataframe
```

```
   profit
a      10
b      20
c      30
d      40
e      50
```

Out[75]:

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

In [76]:

```
print(df6.iloc[[1],:])
type(df6.iloc[[1],:])

# prints as dataframe
```

```
    sp  profit
b  200      20
```

Out[76]:

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

In [77]:

```
print(df6.loc[:,['profit']])
type(df6.loc[:,['profit']])

# prints as dataframe
```

```
   profit
a      10
b      20
c      30
d      40
e      50
```

Out[77]:

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

In [78]:

```
# Create dataframe from Dictionary of lists
```

In [79]:

```
store = ['Walmart','Safeway','Total','Trader_Joe']

sales = [1000,3400,6727,5618]

visitors_per_hr = [139,132,87,73]

Location = ['AL','UT','TX','AR']
```

In [80]:

```
list_labels = ['store','sales','visitors_per_hr','Location']
```

In [81]:

```
list_cols = [store,sales,visitors_per_hr,Location]
```

In [82]:

```
zipped = list(zip(list_labels,list_cols))
zipped
```

Out[82]:

```
[('store', ['Walmart', 'Safeway', 'Total', 'Trader_Joe']),
 ('sales', [1000, 3400, 6727, 5618]),
 ('visitors_per_hr', [139, 132, 87, 73]),
 ('Location', ['AL', 'UT', 'TX', 'AR'])]
```

In [83]:

```
data = dict(zipped)
data
```

Out[83]:

```
{'store': ['Walmart', 'Safeway', 'Total', 'Trader_Joe'],
 'sales': [1000, 3400, 6727, 5618],
 'visitors_per_hr': [139, 132, 87, 73],
 'Location': ['AL', 'UT', 'TX', 'AR']}
```

In [84]:

```
final = DataFrame(data)
final
```

Out[84]:

	store	sales	visitors_per_hr	Location
0	Walmart	1000	139	AL
1	Safeway	3400	132	UT
2	Total	6727	87	TX
3	Trader_Joe	5618	73	AR

In [85]:

```
# concise way

df_final = DataFrame({ 'store' :['Walmart','Safeway','Total','Trader_Joe'], 'sales':[1000,3
                       'visitors_per_hr':[139,132,87,73], 'Location':['AL','UT','TX','AR']
df_final
```

Out[85]:

	store	sales	visitors_per_hr	Location
0	Walmart	1000	139	AL
1	Safeway	3400	132	UT
2	Total	6727	87	TX
3	Trader_Joe	5618	73	AR

In [86]:

```
import os
os.getcwd()

# to get current working directory
```

Out[86]:

```
'C:\\Users\\Swapnil bandekar\\Downloads\\Swapnil\\Data Analytics\\My Work\\P
ython\\2. Introduction to Pandas'
```

In [87]:

```
os.chdir('C:\\Users\\Swapnil bandekar\\Downloads\\Swapnil\\Data Analytics\\My Work\\Python\

# to change current working directory
```

Flat files

- Checking the delimiter
- Checking the header
- Checking how missing values are populated

In [88]:

```
# We will use pd.read_csv() method to import the csv file

dat1 = pd.read_csv('sample2.csv', sep = ',', header=0)
dat1.head()
```

Out[88]:

	Age	Customer	Income	Location
0	30	A	1200.0	A1
1	32	B	1333.0	A2
2	33	NaN	NaN	NaN

In [89]:

```
# To export into csv file will use "to_csv()" method

dat1.to_csv('sample2_exported.csv')
```

In [90]:

```
# We will use pd.read_excel() method to import the XLSX file

dat1_new = pd.read_excel('sample2.xlsx')
dat1_new

# by default first sheet in excel will be imported
```

Out[90]:

	Age	Customer	Income	Location
0	30	A	1200.0	A1
1	32	B	1333.0	A2
2	33	NaN	NaN	NaN

In [92]:

```
# We can specify the sheet name to import the particular sheet from the excel

dat1_new1 = pd.read_excel('sample2.xlsx',sheet_name='Data')
dat1_new1
```

Out[92]:

	Age	Customer	Income	Location
0	45	A	1200.0	A1
1	46	B	1333.0	A2
2	56	NaN	NaN	NaN

In [95]:

```
# To export into xlsx file will use "to_excel()" method

dat1_new.to_excel('sample2_exported.xlsx',sheet_name='exported data',index=False)
```

In [96]:

```
# we can also use pd.read_table() method to do the imports ( for .csv and .txt file )


dat2 = pd.read_table('sample2.csv', sep = ',', header=0)
dat2.head()
```

Out[96]:

	Age	Customer	Income	Location
0	30	A	1200.0	A1
1	32	B	1333.0	A2
2	33	NaN	NaN	NaN

In [97]:

```
dat3 = pd.read_table('sample1.txt', sep = '\t')
dat3.head()

# by default "header = 0"
```

Out[97]:

	Age	Customer	Income	Location
0	30	A	1200	A1
1	32	B	1333	A2
2	33	NaN	Missing	NaN

In [98]:

```
dat2.dtypes
```

Out[98]:

```
Age           int64
Customer     object
Income      float64
Location     object
dtype: object
```

In [99]:

```
dat3 = pd.read_table('sample1.txt', na_values = ['Missing'])
dat3.head()
```

Out[99]:

	Age	Customer	Income	Location
0	30	A	1200.0	A1
1	32	B	1333.0	A2
2	33	NaN	NaN	NaN

In [100]:

```
dat3.dtypes
```

Out[100]:

```
Age           int64
Customer     object
Income      float64
Location     object
dtype: object
```

In [101]:

```
dat4 = pd.read_table('sample1 - copy.txt')
dat4.head()
```

Out[101]:

	Age	Customer	Income	Location
0	30	A	1200	A1
1	32	B	1333	A2
2	33	NaN	Missing	NotAvail

In [102]:

```
dat4 = pd.read_table('sample1 - copy.txt', na_values = ['Missing','NotAvail'])
dat4.head()
```

Out[102]:

	Age	Customer	Income	Location
0	30	A	1200.0	A1
1	32	B	1333.0	A2
2	33	NaN	NaN	NaN

Opening Flat files from the web

- Import 'urlretrieve' from "urllib.request" package

In [103]:

```
import urllib.request
from urllib.request import urlretrieve
```

In [105]:

```
# Assign url to the object url

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-w

# save file locally

urlretrieve( url , 'winequality-white.csv' )
```

Out[105]:

```
('winequality-white.csv', <http.client.HTTPMessage at 0x213bc574848>)
```

In [106]:

```
wq1 = pd.read_csv('winequality-white.csv', sep = ";")
wq1.head()
```

Out[106]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9

In [107]:

```
wq2 = pd.read_table( url , sep = ";")
wq2.head()
```

Out[107]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9

- Open ".data" file from the url

In [110]:

```
# import package

import requests
```

In [111]:

```
# Specify the url as 'url object'

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data'
```

In [112]:

```
# packages the request , send the request and catch the response ( r )

r = requests.get(url)
```

In [113]:

```
# Extract the response

text = r.text

print(text)
```

```
M,0.455,0.365,0.095,0.514,0.2245,0.101,0.15,15
M,0.35,0.265,0.09,0.2255,0.0995,0.0485,0.07,7
F,0.53,0.42,0.135,0.677,0.2565,0.1415,0.21,9
M,0.44,0.365,0.125,0.516,0.2155,0.114,0.155,10
I,0.33,0.255,0.08,0.205,0.0895,0.0395,0.055,7
I,0.425,0.3,0.095,0.3515,0.141,0.0775,0.12,8
F,0.53,0.415,0.15,0.7775,0.237,0.1415,0.33,20
F,0.545,0.425,0.125,0.768,0.294,0.1495,0.26,16
M,0.475,0.37,0.125,0.5095,0.2165,0.1125,0.165,9
F,0.55,0.44,0.15,0.8945,0.3145,0.151,0.32,19
F,0.525,0.38,0.14,0.6065,0.194,0.1475,0.21,14
M,0.43,0.35,0.11,0.406,0.1675,0.081,0.135,10
M,0.49,0.38,0.135,0.5415,0.2175,0.095,0.19,11
F,0.535,0.405,0.145,0.6845,0.2725,0.171,0.205,10
F,0.47,0.355,0.1,0.4755,0.1675,0.0805,0.185,10
M,0.5,0.4,0.13,0.6645,0.258,0.133,0.24,12
I,0.355,0.28,0.085,0.2905,0.095,0.0395,0.115,7
F,0.44,0.34,0.1,0.451,0.188,0.087,0.13,10
M,0.365,0.295,0.08,0.2555,0.097,0.043,0.1,7
```

- Open ".data" file from the url

In [114]:

```
import json

with open('visa.txt') as dat :
    data_json = json.load(dat)
```

In [115]:

```
print(data_json)
```


```
{'fields': [{'id': 'a', 'label': 'COUNTRY', 'type': 'string'}, {'id': 'b',
'label': 'MISSION', 'type': 'string'}, {'id': 'c', 'label': 'VISA ISSUE DA
TE', 'type': 'string'}, {'id': 'd', 'label': 'EMPLOYMENT VISA', 'type': 's
tring'}, {'id': 'e', 'label': 'TOURIST VISA', 'type': 'string'}, {'id':
'f', 'label': 'BUSINESS VISA', 'type': 'string'}, {'id': 'g', 'label': 'CO
NFERENCE VISA', 'type': 'string'}, {'id': 'h', 'label': 'BUS VISA INDSPOUS
E', 'type': 'string'}, {'id': 'i', 'label': 'BUSINESS VISA TRANSFER', 'typ
e': 'string'}, {'id': 'j', 'label': 'DIPLOMATIC VISA', 'type': 'string'},
{'id': 'k', 'label': 'DIPLOMATIC DEPENDANT VISA', 'type': 'string'}, {'i
d': 'l', 'label': 'EMPLOYMENT VISA INDSPOUSE', 'type': 'string'}, {'id':
'm', 'label': 'ENTRY VISA', 'type': 'string'}, {'id': 'n', 'label': 'MEDIC
AL VISA', 'type': 'string'}, {'id': 'o', 'label': 'MEDICAL ATTENDENT VIS
A', 'type': 'string'}, {'id': 'p', 'label': 'MISSIONARY VISA', 'type': 'st
ring'}, {'id': 'q', 'label': 'PILGRIMAGE VISA', 'type': 'string'}, {'id':
'r', 'label': 'RESEARCH VISA', 'type': 'string'}, {'id': 's', 'label': 'PR
OJECT VISA', 'type': 'string'}, {'id': 't', 'label': 'TRANSIT VISA', 'typ
e': 'string'}, {'id': 'u', 'label': 'VISIT VISA', 'type': 'string'}, {'i
d': 'v', 'label': 'ART SURROGACY', 'type': 'string'}, {'id': 'w', 'label':
'LONG TERM VISA TRANSFER', 'type': 'string'}, {'id': 'x', 'label': 'STUDEN
```


In [116]:

```
type(data_json)
```

Out[116]:

```
dict
```

In [117]:

```
data_json.keys()
```

Out[117]:

```
dict_keys(['fields', 'data'])
```

In [118]:

```
pd.DataFrame(data_json['data'], columns = [name['label'] for name in data_json['fields']])
```

Out[118]:

	COUNTRY	MISSION	VISA ISSUE DATE	EMPLOYMENT VISA	TOURIST VISA	BUSINESS VISA	CONFER
0	BANGLADESH	BANGLADESH-CHITTAGONG	01-01-12	0	306	5	
1	BANGLADESH	BANGLADESH-DHAKA	01-01-12	0	1804	10	
2	BANGLADESH	BANGLADESH-RAJSHAHI	01-01-12	0	193	1	
3	UNITED KINGDOM	UK-LONDON	01-01-12	0	4	0	
4	IRAN	IRAN-BANDARABBAS	01-01-12	0	3	0	
...	...	...	...	...	...	...	
17504	UNITED STATES OF AMERICA	USA-WASHINGTON	31-12-12	0	142	13	
17505	VIETNAM	VIETNAM-HANOI	31-12-12	0	6	0	
17506	VIETNAM	VIETNAM-HO-CHI-MINHCITY	31-12-12	0	23	4	
17507	SOUTH AFRICA	SOUTHAFRICA-DURBAN	31-12-12	0	3	0	
17508	SOUTH AFRICA	SOUTHAFRICA-JOHANNESBURG	31-12-12	1	31	6	

17509 rows × 24 columns

- Open google maps api file

In [120]:

```
import urllib.request, json
with urllib.request.urlopen("http://maps.googleapis.com/maps/api/geocode/json?address=googl
    data = json.loads(url.read().decode())
    print(data)
```

```
{'error_message': 'You must use an API key to authenticate each request to G
oogle Maps Platform APIs. For additional information, please refer to htt
p://g.co/dev/maps-no-account', (http://g.co/dev/maps-no-account',) 'result
s': [], 'status': 'REQUEST_DENIED'}
```

output 247

{'status': 'OK', 'results': [{'formatted_address': '1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA', 'address_components': [{'long_name': '1600', 'short_name': '1600', 'types': ['street_number']}, {'long_name': 'Amphitheatre Parkway', 'short_name': 'Amphitheatre Pkwy', 'types': ['route']}, {'long_name': 'Mountain View', 'short_name': 'Mountain View', 'types': ['locality', 'political']}, {'long_name': 'Santa Clara County', 'short_name': 'Santa Clara County', 'types': ['administrative_area_level_2', 'political']}, {'long_name': 'California', 'short_name': 'CA', 'types': ['administrative_area_level_1', 'political']}, {'long_name': 'United States', 'short_name': 'US', 'types': ['country', 'political']}, {'long_name': '94043', 'short_name': '94043', 'types': ['postal_code']}], 'types': ['establishment', 'point_of_interest'], 'place_id': 'ChIJj61dQgK6j4AR4GeTYWZsKWw', 'geometry': {'location_type': 'ROOFTOP', 'viewport': {'northeast': {'lng': -122.0827085197085, 'lat': 37.4233488802915}, 'southwest': {'lng': -122.0854064802915, 'lat': 37.4206509197085}}, 'location': {'lng': -122.0840575, 'lat': 37.4219999}}}]}

Working with DataFrames

In [125]:

```
df7 = pd.read_excel('3_NEWS_Sales.xlsx',sheet_name='NEWS')
df7
```

Out[125]:

	Month	North	East	West	South
0	Jan	5143	2027	7256	7428
1	Feb	9492	3506	7047	8374
2	Mar	6223	1419	7866	3485
3	Apr	6537	2986	7062	8061
4	May	9186	4614	5657	9003
5	Jun	6999	8625	5130	2725
6	Jul	4882	2341	7418	8768
7	Aug	7762	5923	6715	3117
8	Sep	9629	6063	4306	8304
9	Oct	2239	5743	5530	3131
10	Nov	5232	5784	6924	3996
11	Dec	1455	4896	3557	1005

In [126]:

```
# Setting index


df7.set_index('Month',inplace=True,drop=True)
df7
```

Out[126]:

Month	North	East	West	South
Jan	5143	2027	7256	7428
Feb	9492	3506	7047	8374
Mar	6223	1419	7866	3485
Apr	6537	2986	7062	8061
May	9186	4614	5657	9003
Jun	6999	8625	5130	2725
Jul	4882	2341	7418	8768
Aug	7762	5923	6715	3117
Sep	9629	6063	4306	8304
Oct	2239	5743	5530	3131
Nov	5232	5784	6924	3996
Dec	1455	4896	3557	1005

Selection and Indexing

In [127]:

```
df7['North']
```

Out[127]:

```
Month
Jan    5143
Feb    9492
Mar    6223
Apr    6537
May    9186
Jun    6999
Jul    4882
Aug    7762
Sep    9629
Oct    2239
Nov    5232
Dec    1455
Name: North, dtype: int64
```

In [128]:

```
df7[['North','South']]
```

Out[128]:

Month	North	South
Jan	5143	7428
Feb	9492	8374
Mar	6223	3485
Apr	6537	8061
May	9186	9003
Jun	6999	2725
Jul	4882	8768
Aug	7762	3117
Sep	9629	8304
Oct	2239	3131
Nov	5232	3996
Dec	1455	1005

In [133]:

```
df7.loc['Jan']
```

Out[133]:

```
North    5143
East     2027
West     7256
South    7428
Name: Jan, dtype: int64
```

In [134]:

```
# Creating a new column

df7['North_South'] = df7['North'] + df7['South']
```

In [135]:

```
df7
```

Out[135]:

Month	North	East	West	South	North_South
Jan	5143	2027	7256	7428	12571
Feb	9492	3506	7047	8374	17866
Mar	6223	1419	7866	3485	9708
Apr	6537	2986	7062	8061	14598
May	9186	4614	5657	9003	18189
Jun	6999	8625	5130	2725	9724
Jul	4882	2341	7418	8768	13650
Aug	7762	5923	6715	3117	10879
Sep	9629	6063	4306	8304	17933
Oct	2239	5743	5530	3131	5370
Nov	5232	5784	6924	3996	9228
Dec	1455	4896	3557	1005	2460

In [136]:

```
# Deleting a column

df7.drop('North_South', axis=1, inplace=True)
df7
```

Out[136]:

Month	North	East	West	South
Jan	5143	2027	7256	7428
Feb	9492	3506	7047	8374
Mar	6223	1419	7866	3485
Apr	6537	2986	7062	8061
May	9186	4614	5657	9003
Jun	6999	8625	5130	2725
Jul	4882	2341	7418	8768
Aug	7762	5923	6715	3117
Sep	9629	6063	4306	8304
Oct	2239	5743	5530	3131
Nov	5232	5784	6924	3996
Dec	1455	4896	3557	1005

In [139]:

```
df7.drop(['North','South'], axis=1)
```

Out[139]:

Month	East	West
Jan	2027	7256
Feb	3506	7047
Mar	1419	7866
Apr	2986	7062
May	4614	5657
Jun	8625	5130
Jul	2341	7418
Aug	5923	6715
Sep	6063	4306
Oct	5743	5530
Nov	5784	6924
Dec	4896	3557

In [140]:

```
# Deleting a row

df7.drop('Jan', axis=0)
```

Out[140]:

Month	North	East	West	South
Feb	9492	3506	7047	8374
Mar	6223	1419	7866	3485
Apr	6537	2986	7062	8061
May	9186	4614	5657	9003
Jun	6999	8625	5130	2725
Jul	4882	2341	7418	8768
Aug	7762	5923	6715	3117
Sep	9629	6063	4306	8304
Oct	2239	5743	5530	3131
Nov	5232	5784	6924	3996
Dec	1455	4896	3557	1005

Selection and Indexing

In [127]:

```
df7['North']
```

Out[127]:

```
Month
Jan    5143
Feb    9492
Mar    6223
Apr    6537
May    9186
Jun    6999
Jul    4882
Aug    7762
Sep    9629
Oct    2239
Nov    5232
Dec    1455
Name: North, dtype: int64
```

In [128]:

```
df7[['North','South']]
```

Out[128]:

Month	North	South
Jan	5143	7428
Feb	9492	8374
Mar	6223	3485
Apr	6537	8061
May	9186	9003
Jun	6999	2725
Jul	4882	8768
Aug	7762	3117
Sep	9629	8304
Oct	2239	3131
Nov	5232	3996
Dec	1455	1005

In [133]:

```
# Selecting Rows

df7.loc['Jan']
```

Out[133]:

```
North    5143
East     2027
West     7256
South    7428
Name: Jan, dtype: int64
```

In [141]:

```
df7.iloc[2]
```

Out[141]:

```
North    6223
East     1419
West     7866
South    3485
Name: Mar, dtype: int64
```

In [142]:

```
# Selecting subset of rows and columns

df7.loc['Mar','South']
```

Out[142]:

```
3485
```

In [144]:

```
df7.loc[['Mar','Apr'],['North','West']]
```

Out[144]:

Month	North	West
Mar	6223	7866
Apr	6537	7062

Conditional Selection

- An important feature of pandas is conditional selection using bracket notation

In [145]:

```
df7
```

Out[145]:

Month	North	East	West	South
Jan	5143	2027	7256	7428
Feb	9492	3506	7047	8374
Mar	6223	1419	7866	3485
Apr	6537	2986	7062	8061
May	9186	4614	5657	9003
Jun	6999	8625	5130	2725
Jul	4882	2341	7418	8768
Aug	7762	5923	6715	3117
Sep	9629	6063	4306	8304
Oct	2239	5743	5530	3131
Nov	5232	5784	6924	3996
Dec	1455	4896	3557	1005

In [146]:

```
df7>8000
```

Out[146]:

Month	North	East	West	South
Jan	False	False	False	False
Feb	True	False	False	True
Mar	False	False	False	False
Apr	False	False	False	True
May	True	False	False	True
Jun	False	True	False	False
Jul	False	False	False	True
Aug	False	False	False	False
Sep	True	False	False	True
Oct	False	False	False	False
Nov	False	False	False	False
Dec	False	False	False	False

In [147]:

```
df7[df7>8000]
```

Out[147]:

Month	North	East	West	South
Jan	NaN	NaN	NaN	NaN
Feb	9492.0	NaN	NaN	8374.0
Mar	NaN	NaN	NaN	NaN
Apr	NaN	NaN	NaN	8061.0
May	9186.0	NaN	NaN	9003.0
Jun	NaN	8625.0	NaN	NaN
Jul	NaN	NaN	NaN	8768.0
Aug	NaN	NaN	NaN	NaN
Sep	9629.0	NaN	NaN	8304.0
Oct	NaN	NaN	NaN	NaN
Nov	NaN	NaN	NaN	NaN
Dec	NaN	NaN	NaN	NaN

In [149]:

```
df7[df7['North']>8000]
```

Out[149]:

Month	North	East	West	South
Feb	9492	3506	7047	8374
May	9186	4614	5657	9003
Sep	9629	6063	4306	8304

In [150]:

```
df7[df7['North']>8000]['East']
```

Out[150]:

```
Month
Feb    3506
May    4614
Sep    6063
Name: East, dtype: int64
```

In [152]:

```
df7[df7['North']>8000][['East','West']]
```

Out[152]:

Month	East	West
Feb	3506	7047
May	4614	5657
Sep	6063	4306

For two conditions , we can use | (OR) and & (AND) with parenthesis

In [155]:

```
df7[ (df7['North']>8000) & (df7['East']>5000) ]
```

Out[155]:

Month	North	East	West	South
Sep	9629	6063	4306	8304

In [159]:

```
df7[ (df7['North']>8000) | (df7['East']>7000) ]
```

Out[159]:

Month	North	East	West	South
Feb	9492	3506	7047	8374
May	9186	4614	5657	9003
Jun	6999	8625	5130	2725
Sep	9629	6063	4306	8304

In [160]:

```
df7
```

Out[160]:

Month	North	East	West	South
Jan	5143	2027	7256	7428
Feb	9492	3506	7047	8374
Mar	6223	1419	7866	3485
Apr	6537	2986	7062	8061
May	9186	4614	5657	9003
Jun	6999	8625	5130	2725
Jul	4882	2341	7418	8768
Aug	7762	5923	6715	3117
Sep	9629	6063	4306	8304
Oct	2239	5743	5530	3131
Nov	5232	5784	6924	3996
Dec	1455	4896	3557	1005

More Index Details

- Let's discuss some more features of indexing, including resetting the index or setting it something else. Also, about index hierarchy!

In [161]:

```
# Reseting the index to default

df7.reset_index()
```

Out[161]:

	Month	North	East	West	South
0	Jan	5143	2027	7256	7428
1	Feb	9492	3506	7047	8374
2	Mar	6223	1419	7866	3485
3	Apr	6537	2986	7062	8061
4	May	9186	4614	5657	9003
5	Jun	6999	8625	5130	2725
6	Jul	4882	2341	7418	8768
7	Aug	7762	5923	6715	3117
8	Sep	9629	6063	4306	8304
9	Oct	2239	5743	5530	3131
10	Nov	5232	5784	6924	3996
11	Dec	1455	4896	3557	1005

In [ ]:

```
# Adding a new index column
```

In [163]:

```
newindex = 'Jan19 Feb19 Mar19 Apr19 May19 Jun19 Jul19 Aug19 Sep19 Oct19 Nov19 Dec19'.split(
```

In [164]:

```
newindex
```

Out[164]:

```
['Jan19',
 'Feb19',
 'Mar19',
 'Apr19',
 'May19',
 'Jun19',
 'Jul19',
 'Aug19',
 'Sep19',
 'Oct19',
 'Nov19',
 'Dec19']
```

In [165]:

```
df7['Year19'] = newindex
df7
```

Out[165]:

Month	North	East	West	South	Year19
Jan	5143	2027	7256	7428	Jan19
Feb	9492	3506	7047	8374	Feb19
Mar	6223	1419	7866	3485	Mar19
Apr	6537	2986	7062	8061	Apr19
May	9186	4614	5657	9003	May19
Jun	6999	8625	5130	2725	Jun19
Jul	4882	2341	7418	8768	Jul19
Aug	7762	5923	6715	3117	Aug19
Sep	9629	6063	4306	8304	Sep19
Oct	2239	5743	5530	3131	Oct19
Nov	5232	5784	6924	3996	Nov19
Dec	1455	4896	3557	1005	Dec19

In [168]:

```
df7.set_index('Year19', inplace= True )
```

In [169]:

```
df7
```

Out[169]:

Year19	North	East	West	South
Jan19	5143	2027	7256	7428
Feb19	9492	3506	7047	8374
Mar19	6223	1419	7866	3485
Apr19	6537	2986	7062	8061
May19	9186	4614	5657	9003
Jun19	6999	8625	5130	2725
Jul19	4882	2341	7418	8768
Aug19	7762	5923	6715	3117
Sep19	9629	6063	4306	8304
Oct19	2239	5743	5530	3131
Nov19	5232	5784	6924	3996
Dec19	1455	4896	3557	1005

Multi-Index and Index Hierarchy

- Let's go over how to work with Multi-Index, first we'll create a quick example of what a Multi-Indexed DataFrame would look like:

In [178]:

```
L1 = ['G1','G1','G1','G2','G2','G2']
L2 = [1,2,3,1,2,3]
Hier_Index1 = list(zip(L1,L2))
Hier_Index = pd.MultiIndex.from_tuples(Hier_Index1)
```

In [179]:

```
Hier_Index
```

Out[179]:

```
MultiIndex([('G1', 1),
            ('G1', 2),
            ('G1', 3),
            ('G2', 1),
            ('G2', 2),
            ('G2', 3)],
           )
```

In [183]:

```
df8 = pd.DataFrame(data = np.random.rand(6,2), index = Hier_Index, columns = ['A','B'])
df8
```

Out[183]:

		A	B
G1	1	0.370009	0.606958
	2	0.848485	0.591374
	3	0.035400	0.121174
G2	1	0.193969	0.950578
	2	0.164425	0.052655
	3	0.642101	0.543481

In [185]:

```
df8.loc['G1']
```

Out[185]:

	A	B
1	0.370009	0.606958
2	0.848485	0.591374
3	0.035400	0.121174

In [191]:

```
df8.loc['G1'].loc[1]
```

Out[191]:

```
A    0.370009
B    0.606958
Name: 1, dtype: float64
```

In [192]:

```
df8.index.names
```

Out[192]:

```
FrozenList([None, None])
```

In [193]:

```
df8.index.names = ['Group','Sub-Group']
```

In [194]:

```
df8
```

Out[194]:

		A	B
Group	**Sub-Group**		
G1	**1**	0.370009	0.606958
	2	0.848485	0.591374
	3	0.035400	0.121174
G2	**1**	0.193969	0.950578
	2	0.164425	0.052655
	3	0.642101	0.543481

In [195]:

```
df8.xs('G1')
```

Out[195]:

	A	B
Sub-Group		
1	0.370009	0.606958
2	0.848485	0.591374
3	0.035400	0.121174

In [196]:

```
df8.xs(['G1',1])
```

Out[196]:

```
A    0.370009
B    0.606958
Name: (G1, 1), dtype: float64
```

In [197]:

```
df8.xs(1,level='Sub-Group')
```

Out[197]:

	A	B
Group		
G1	0.370009	0.606958
G2	0.193969	0.950578

In [199]:

```
mul = pd.MultiIndex(levels = [['G1','G2'],[1,2,3]],
                    codes = [[0,0,0,1,1,1],[0,1,2,0,1,2]])
```

In [203]:

```
df8_copy = pd.DataFrame(data = np.random.rand(6,2) , index = mul , columns = ['A','B'])
df8_copy
```

Out[203]:

		A	B
G1	1	0.114646	0.060890
	2	0.877005	0.078392
	3	0.593029	0.799450
G2	1	0.834839	0.782842
	2	0.166932	0.166299
	3	0.221166	0.479915

In [206]:

```
df8_copy.index.names = ['Group','Sub-Group']
df8_copy
```

Out[206]:

		A	B
Group	Sub-Group		
G1	1	0.114646	0.060890
	2	0.877005	0.078392
	3	0.593029	0.799450
G2	1	0.834839	0.782842
	2	0.166932	0.166299
	3	0.221166	0.479915

Missing Data Treatment

In [209]:

```
df9 = pd.DataFrame({'A':[1,2,np.nan],
                    'B':[1,np.nan,np.nan],
                    'C':[1,2,3]})
df9
```

Out[209]:

	A	B	C
0	1.0	1.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [210]:

```
df9.dropna()
```

Out[210]:

	A	B	C
0	1.0	1.0	1

In [213]:

```
df9.dropna(axis=1)
```

Out[213]:

	C
0	1
1	2
2	3

In [216]:

```
df9.dropna(thresh=2)

# tresh = 2 : Alleast 2 values in a row should have 'non NaN' value
```

Out[216]:

	A	B	C
0	1.0	1.0	1
1	2.0	NaN	2

In [217]:

```
df9.fillna(value='Missing')
```

Out[217]:

	A	B	C
0	1	1	1
1	2	Missing	2
2	Missing	Missing	3

In [218]:

```
df9['A'].fillna(value = df9['A'].mean())
```

Out[218]:

```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

In [220]:

```
df10 = pd.DataFrame({'A':[1,2,np.nan,4,5,np.nan,7,8,9,np.nan],
                     'B':[1,np.nan,np.nan,4,5,6,np.nan,8,np.nan,10],
                     'C':[np.nan,2,3,np.nan,np.nan,np.nan,7,8,np.nan,np.nan]})
df10
```

Out[220]:

	A	B	C
0	1.0	1.0	NaN
1	2.0	NaN	2.0
2	NaN	NaN	3.0
3	4.0	4.0	NaN
4	5.0	5.0	NaN
5	NaN	6.0	NaN
6	7.0	NaN	7.0
7	8.0	8.0	8.0
8	9.0	NaN	NaN
9	NaN	10.0	NaN

In [225]:

```
df10.fillna(method='bfill')
```

Out[225]:

	A	B	C
0	1.0	1.0	2.0
1	2.0	4.0	2.0
2	4.0	4.0	3.0
3	4.0	4.0	7.0
4	5.0	5.0	7.0
5	7.0	6.0	7.0
6	7.0	8.0	7.0
7	8.0	8.0	8.0
8	9.0	10.0	NaN
9	NaN	10.0	NaN

In [226]:

```
df10.fillna(method='pad')
```

Out[226]:

	A	B	C
0	1.0	1.0	NaN
1	2.0	1.0	2.0
2	2.0	1.0	3.0
3	4.0	4.0	3.0
4	5.0	5.0	3.0
5	5.0	6.0	3.0
6	7.0	6.0	7.0
7	8.0	8.0	8.0
8	9.0	8.0	8.0
9	9.0	10.0	8.0

Groupby

- We can summarize the data by using the Groupby Method

In [229]:

```
df11 = pd.read_csv('5_Sales.csv')
df11.head()
```

Out[229]:

	SalesRep	Region	Month	Sales	Units Sold
0	Amy	North	Jan	23040	239
1	Amy	North	Feb	24131	79
2	Amy	North	Mar	24646	71
3	Amy	North	Apr	22047	71
4	Amy	North	May	24971	80

In [230]:

```
# Summarize by SalesRep

df11.groupby('SalesRep')
```

Out[230]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000213C18BFB08>
```

In [237]:

```
# Saving the result of groupby method into and object

data1 = df11.groupby('SalesRep')
```

In [235]:

```
data1 = pd.DataFrame(df11.groupby('SalesRep'))
```

In [236]:

```
data1
```

Out[236]:

	0	1
0	Amy	SalesRep Region Month Sales Units Sold 0 ...
1	Bob	SalesRep Region Month Sales Units Sold 12...
2	Chuck	SalesRep Region Month Sales Units Sold 24...
3	Doug	SalesRep Region Month Sales Units Sold 36...

In [241]:

```
# Calling aggregate methods of the object

data1.mean()
```

Out[241]:

SalesRep	Sales	Units Sold
Amy	24575.50	202.666667
Bob	23476.25	169.583333
Chuck	21453.00	250.333333
Doug	27372.00	193.333333

In [240]:

```
df11.groupby('SalesRep')dd
```

Out[240]:

SalesRep	Sales	Units Sold
Amy	24575.50	202.666667
Bob	23476.25	169.583333
Chuck	21453.00	250.333333
Doug	27372.00	193.333333

In [246]:

```
df11.groupby('SalesRep').mean()
```

Out[246]:

SalesRep	Sales	Units Sold
Amy	24575.50	202.666667
Bob	23476.25	169.583333
Chuck	21453.00	250.333333
Doug	27372.00	193.333333

In [247]:

```
data1.max()
```

Out[247]:

SalesRep	Region	Month	Sales	Units Sold
Amy	North	Sep	25899	706
Bob	North	Sep	25999	465
Chuck	South	Sep	23965	769
Doug	South	Sep	29953	852

In [248]:

```
data1.min()
```

Out[248]:

SalesRep	Region	Month	Sales	Units Sold
Amy	North	Apr	22047	71
Bob	North	Apr	20024	68
Chuck	South	Apr	19625	70
Doug	South	Apr	25041	81

In [249]:

```
data1.std()
```

Out[249]:

SalesRep	Sales	Units Sold
Amy	1142.101452	209.702531
Bob	1706.800203	117.599829
Chuck	1479.552635	261.063675
Doug	1876.092021	215.187586

In [250]:

```
data1.count()
```

Out[250]:

SalesRep	Region	Month	Sales	Units Sold
Amy	12	12	12	12
Bob	12	12	12	12
Chuck	12	12	12	12
Doug	12	12	12	12

In [251]:

```
data1.describe()
```

Out[251]:

								Sales	
SalesRep	count	mean	std	min	25%	50%	75%	max	count
Amy	12.0	24575.50	1142.101452	22047.0	24196.25	24662.0	25450.00	25899.0	12.0
Bob	12.0	23476.25	1706.800203	20024.0	22788.25	23500.5	24763.25	25999.0	12.0
Chuck	12.0	21453.00	1479.552635	19625.0	20181.50	21428.5	22301.50	23965.0	12.0
Doug	12.0	27372.00	1876.092021	25041.0	25752.25	27242.0	29130.25	29953.0	12.0

In [252]:

```
data1.describe().transpose()
```

Out[252]:

	SalesRep	Amy	Bob	Chuck	Doug
Sales	count	12.000000	12.000000	12.000000	12.000000
	mean	24575.500000	23476.250000	21453.000000	27372.000000
	std	1142.101452	1706.800203	1479.552635	1876.092021
	min	22047.000000	20024.000000	19625.000000	25041.000000
	25%	24196.250000	22788.250000	20181.500000	25752.250000
	50%	24662.000000	23500.500000	21428.500000	27242.000000
	75%	25450.000000	24763.250000	22301.500000	29130.250000
	max	25899.000000	25999.000000	23965.000000	29953.000000
Units Sold	count	12.000000	12.000000	12.000000	12.000000
	mean	202.666667	169.583333	250.333333	193.333333
	std	209.702531	117.599829	261.063675	215.187586
	min	71.000000	68.000000	70.000000	81.000000
	25%	79.750000	93.000000	92.000000	90.500000
	50%	93.500000	111.500000	131.000000	111.500000
	75%	194.750000	238.250000	246.750000	204.250000
	max	706.000000	465.000000	769.000000	852.000000

In [253]:

```
data1.describe().transpose()['Amy']
```

Out[253]:

```
Sales       count       12.000000
            mean     24575.500000
            std       1142.101452
            min      22047.000000
            25%      24196.250000
            50%      24662.000000
            75%      25450.000000
            max      25899.000000
Units Sold  count       12.000000
            mean       202.666667
            std        209.702531
            min         71.000000
            25%         79.750000
            50%         93.500000
            75%        194.750000
            max        706.000000
Name: Amy, dtype: float64
```

Merging, Joining, and Concatenating

- There are 3 main ways of combining DataFrames together: Merging, Joining and Concatenating.

In [262]:

```
df1_c =pd.read_csv('6_concatenate_df1.csv')
df2_c =pd.read_csv('6_concatenate_df2.csv')
df3_c =pd.read_csv('6_concatenate_df3.csv')
```

In [264]:

```
df1_c
```

Out[264]:

	ID	Student_Name	Score
0	101	Manoj	12
1	102	Rakesh	68
2	103	Rosy	87
3	104	Yogesh	95

In [265]:

```
df2_c
```

Out[265]:

	ID	Student_Name	Score
0	105	Rahul	69
1	106	Anu	73
2	107	Rosalin	5
3	108	Ankur	13

In [266]:

```
df3_c
```

Out[266]:

	ID	Student_Name	Score
0	109	Arvind	51
1	110	Aradhana	57
2	111	Avinash	79
3	112	Deepthi	28

Concatenation

- Concatenation basically glues together DataFrames. Keep in mind that dimensions should match along the axis you are concatenating on. You can use **pd.concat** and pass in a list of DataFrames to concatenate together:

In [269]:

```
pd.concat([df1_c,df2_c,df3_c])
```

Out[269]:

	ID	Student_Name	Score
0	101	Manoj	12
1	102	Rakesh	68
2	103	Rosy	87
3	104	Yogesh	95
0	105	Rahul	69
1	106	Anu	73
2	107	Rosalin	5
3	108	Ankur	13
0	109	Arvind	51
1	110	Aradhana	57
2	111	Avinash	79
3	112	Deepthi	28

Merging

- The **merge** function allows you to merge DataFrames together using a similar logic as merging SQL Tables together.

In [270]:

```
df1_m = pd.read_csv('6_merge_df1.csv')
df2_m = pd.read_csv('6_merge_df2.csv')
```

In [271]:

```
df1_m
```

Out[271]:

	Cust_ID	Customer
0	101	Manoj
1	102	Rakesh
2	103	Rosy
3	104	Yogesh

In [272]:

```
df1_m.drop(0,axis=0,inplace=True)
```

In [273]:

```
df1_m
```

Out[273]:

	Cust_ID	Customer
1	102	Rakesh
2	103	Rosy
3	104	Yogesh

In [275]:

```
df2_m
```

Out[275]:

	Purchase_ID	Cust_ID	Sales
0	101	103	6235
1	102	104	7533
2	103	102	9427
3	104	101	5795
4	105	101	4486
5	106	102	2750
6	107	101	4453
7	108	104	6048
8	109	101	2915
9	110	102	4103

In [276]:

```
pd.merge( df1_m, df2_m, how = 'left' , on = 'Cust_ID' )
```

Out[276]:

	Cust_ID	Customer	Purchase_ID	Sales
0	102	Rakesh	103	9427
1	102	Rakesh	106	2750
2	102	Rakesh	110	4103
3	103	Rosy	101	6235
4	104	Yogesh	102	7533
5	104	Yogesh	108	6048

In [277]:

```
pd.merge( df1_m, df2_m, how = 'right' , on = 'Cust_ID' )
```

Out[277]:

	Cust_ID	Customer	Purchase_ID	Sales
0	102	Rakesh	103	9427
1	102	Rakesh	106	2750
2	102	Rakesh	110	4103
3	103	Rosy	101	6235
4	104	Yogesh	102	7533
5	104	Yogesh	108	6048
6	101	NaN	104	5795
7	101	NaN	105	4486
8	101	NaN	107	4453
9	101	NaN	109	2915

In [278]:

```
pd.merge( df2_m, df1_m, how = 'left' , on = 'Cust_ID' )
```

Out[278]:

	Purchase_ID	Cust_ID	Sales	Customer
0	101	103	6235	Rosy
1	102	104	7533	Yogesh
2	103	102	9427	Rakesh
3	104	101	5795	NaN
4	105	101	4486	NaN
5	106	102	2750	Rakesh
6	107	101	4453	NaN
7	108	104	6048	Yogesh
8	109	101	2915	NaN
9	110	102	4103	Rakesh

In [279]:

```
pd.merge( df2_m, df1_m, how = 'right' , on = 'Cust_ID' )
```

Out[279]:

	Purchase_ID	Cust_ID	Sales	Customer
0	101	103	6235	Rosy
1	102	104	7533	Yogesh
2	108	104	6048	Yogesh
3	103	102	9427	Rakesh
4	106	102	2750	Rakesh
5	110	102	4103	Rakesh

Joining

- Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.

In [282]:

```
df1_j = pd.read_csv('6_merge_df1.csv')
df2_j = pd.read_csv('6_merge_df2.csv')
```

In [283]:

```
df1_j
```

Out[283]:

	Cust_ID	Customer
0	101	Manoj
1	102	Rakesh
2	103	Rosy
3	104	Yogesh

In [284]:

```
df2_j
```

Out[284]:

	Purchase_ID	Cust_ID	Sales
0	101	103	6235
1	102	104	7533
2	103	102	9427
3	104	101	5795
4	105	101	4486
5	106	102	2750
6	107	101	4453
7	108	104	6048
8	109	101	2915
9	110	102	4103

In [285]:

```
df2_j.set_index('Cust_ID',inplace=True)
```

In [286]:

```
df2_j
```

Out[286]:

Cust_ID	Purchase_ID	Sales
103	101	6235
104	102	7533
102	103	9427
101	104	5795
101	105	4486
102	106	2750
101	107	4453
104	108	6048
101	109	2915
102	110	4103

In [289]:

```
df1_j.join(df2_j)
```

Out[289]:

	Cust_ID	Customer	Purchase_ID	Sales
0	101	Manoj	NaN	NaN
1	102	Rakesh	NaN	NaN
2	103	Rosy	NaN	NaN
3	104	Yogesh	NaN	NaN

In [290]:

```
df1_j.join(df2_j,on='Cust_ID')
```

Out[290]:

	Cust_ID	Customer	Purchase_ID	Sales
0	101	Manoj	104	5795
0	101	Manoj	105	4486
0	101	Manoj	107	4453
0	101	Manoj	109	2915
1	102	Rakesh	103	9427
1	102	Rakesh	106	2750
1	102	Rakesh	110	4103
2	103	Rosy	101	6235
3	104	Yogesh	102	7533
3	104	Yogesh	108	6048

In [291]:

```
df1_j.join(df2_j,on='Cust_ID',how='right')
```

Out[291]:

	Cust_ID	Customer	Purchase_ID	Sales
0	101	Manoj	104	5795
0	101	Manoj	105	4486
0	101	Manoj	107	4453
0	101	Manoj	109	2915
1	102	Rakesh	103	9427
1	102	Rakesh	106	2750
1	102	Rakesh	110	4103
2	103	Rosy	101	6235
3	104	Yogesh	102	7533
3	104	Yogesh	108	6048

In [292]:

```
df1_j.join(df2_j,on='Cust_ID',how='inner')
```

Out[292]:

	Cust_ID	Customer	Purchase_ID	Sales
0	101	Manoj	104	5795
0	101	Manoj	105	4486
0	101	Manoj	107	4453
0	101	Manoj	109	2915
1	102	Rakesh	103	9427
1	102	Rakesh	106	2750
1	102	Rakesh	110	4103
2	103	Rosy	101	6235
3	104	Yogesh	102	7533
3	104	Yogesh	108	6048

In [293]:

```
df1_j.join(df2_j,on='Cust_ID',how='outer')
```

Out[293]:

	Cust_ID	Customer	Purchase_ID	Sales
0	101	Manoj	104	5795
0	101	Manoj	105	4486
0	101	Manoj	107	4453
0	101	Manoj	109	2915
1	102	Rakesh	103	9427
1	102	Rakesh	106	2750
1	102	Rakesh	110	4103
2	103	Rosy	101	6235
3	104	Yogesh	102	7533
3	104	Yogesh	108	6048

In [294]:

```
df2_j
```

Out[294]:

Cust_ID	Purchase_ID	Sales
103	101	6235
104	102	7533
102	103	9427
101	104	5795
101	105	4486
102	106	2750
101	107	4453
104	108	6048
101	109	2915
102	110	4103

In [295]:

```
df1_j
```

Out[295]:

	Cust_ID	Customer
0	101	Manoj
1	102	Rakesh
2	103	Rosy
3	104	Yogesh

In [296]:

```
df1_j.set_index('Cust_ID',inplace=True)
df1_j
```

Out[296]:

	Customer
Cust_ID	
101	Manoj
102	Rakesh
103	Rosy
104	Yogesh

In [297]:

```
df1_j.join(df2_j)
```

Out[297]:

	Customer	Purchase_ID	Sales
Cust_ID			
101	Manoj	104	5795
101	Manoj	105	4486
101	Manoj	107	4453
101	Manoj	109	2915
102	Rakesh	103	9427
102	Rakesh	106	2750
102	Rakesh	110	4103
103	Rosy	101	6235
104	Yogesh	102	7533
104	Yogesh	108	6048

In [299]:

```
df1_j.join(df2_j,how='right')
```

Out[299]:

Cust_ID	Customer	Purchase_ID	Sales
101	Manoj	104	5795
101	Manoj	105	4486
101	Manoj	107	4453
101	Manoj	109	2915
102	Rakesh	103	9427
102	Rakesh	106	2750
102	Rakesh	110	4103
103	Rosy	101	6235
104	Yogesh	102	7533
104	Yogesh	108	6048

In [300]:

```
df1_j.join(df2_j,how='inner')
```

Out[300]:

Cust_ID	Customer	Purchase_ID	Sales
101	Manoj	104	5795
101	Manoj	105	4486
101	Manoj	107	4453
101	Manoj	109	2915
102	Rakesh	103	9427
102	Rakesh	106	2750
102	Rakesh	110	4103
103	Rosy	101	6235
104	Yogesh	102	7533
104	Yogesh	108	6048

In [301]:

```
df1_j.join(df2_j,how='outer')
```

Out[301]:

Cust_ID	Customer	Purchase_ID	Sales
101	Manoj	104	5795
101	Manoj	105	4486
101	Manoj	107	4453
101	Manoj	109	2915
102	Rakesh	103	9427
102	Rakesh	106	2750
102	Rakesh	110	4103
103	Rosy	101	6235
104	Yogesh	102	7533
104	Yogesh	108	6048

In [302]:

```
df1_j.join(df2_j)
```

Out[302]:

Cust_ID	Customer	Purchase_ID	Sales
101	Manoj	104	5795
101	Manoj	105	4486
101	Manoj	107	4453
101	Manoj	109	2915
102	Rakesh	103	9427
102	Rakesh	106	2750
102	Rakesh	110	4103
103	Rosy	101	6235
104	Yogesh	102	7533
104	Yogesh	108	6048

In [303]:

```
df2_j.join(df1_j)
```

Out[303]:

Cust_ID	Purchase_ID	Sales	Customer
101	104	5795	Manoj
101	105	4486	Manoj
101	107	4453	Manoj
101	109	2915	Manoj
102	103	9427	Rakesh
102	106	2750	Rakesh
102	110	4103	Rakesh
103	101	6235	Rosy
104	102	7533	Yogesh
104	108	6048	Yogesh

Pandas Operations

- There are lots of operations with pandas that will be really useful to us

In [306]:

```
df12 = pd.read_csv('1_Sales.csv')
df12.head()
```

Out[306]:

	SalesRep	Region	Month	Sales	Units Sold
0	Amy	North	Jan	23040.0	239.0
1	Amy	North	Feb	24131.0	79.0
2	Amy	North	Mar	24646.0	71.0
3	Amy	North	Apr	22047.0	71.0
4	Amy	North	May	24971.0	NaN

In [307]:

```
df12.describe()
```

Out[307]:

	Sales	Units Sold
count	47.000000	47.000000
mean	24209.425532	206.617021
std	2671.207948	204.332269
min	19625.000000	68.000000
25%	22348.500000	86.500000
50%	24218.000000	114.000000
75%	25742.000000	227.000000
max	29953.000000	852.000000

In [312]:

```
# Getting Unique Values

df12['SalesRep'].unique()
```

Out[312]:

```
array(['Amy', 'Bob', 'Chuck', 'Doug'], dtype=object)
```

In [313]:

```
df12['SalesRep'].nunique()
```

Out[313]:

```
4
```

In [314]:

```
df12['SalesRep'].value_counts()
```

Out[314]:

```
Doug     12
Chuck    12
Amy      12
Bob      12
Name: SalesRep, dtype: int64
```

In [316]:

```
# Selecting data from multiple columns

df12[(df12['Month']=='Jan') & (df12['Sales']>24000)]
```

Out[316]:

	SalesRep	Region	Month	Sales	Units Sold
36	Doug	South	Jan	26264.0	92.0

Applying funtions to the DataFrames

In [320]:

```
def times2(x):
    return x*2
```

In [321]:

```
df12['Units Sold'].apply(times2).head()
```

Out[321]:

```
0    478.0
1    158.0
2    142.0
3    142.0
4      NaN
Name: Units Sold, dtype: float64
```

In [322]:

```
df12['SalesRep'].apply(len).head()
```

Out[322]:

```
0    3
1    3
2    3
3    3
4    3
Name: SalesRep, dtype: int64
```

In [323]:

```
def compute_AUP(cols):
    aup1 = cols[0]
    aup2 = cols[1]
    return aup1/aup2
```

In [327]:

```
df12['AUP'] = df12[['Sales','Units Sold']].apply(compute_AUP , axis=1)
```

In [328]:

```
df12.head()
```

Out[328]:

	SalesRep	Region	Month	Sales	Units Sold	AUP
0	Amy	North	Jan	23040.0	239.0	96.401674
1	Amy	North	Feb	24131.0	79.0	305.455696
2	Amy	North	Mar	24646.0	71.0	347.126761
3	Amy	North	Apr	22047.0	71.0	310.521127
4	Amy	North	May	24971.0	NaN	NaN

In [329]:

```
df12['Sales'].sum()
```

Out[329]:

```
1137843.0
```

In [333]:

```
# Permanently removing the column from DataFrame

del df12['AUP']
df12.head()
```

Out[333]:

	SalesRep	Region	Month	Sales	Units Sold
0	Amy	North	Jan	23040.0	239.0
1	Amy	North	Feb	24131.0	79.0
2	Amy	North	Mar	24646.0	71.0
3	Amy	North	Apr	22047.0	71.0
4	Amy	North	May	24971.0	NaN

In [334]:

```
# Getting column names

df12.columns
```

Out[334]:

```
Index(['SalesRep', 'Region', 'Month', 'Sales', 'Units Sold'], dtype='objec
t')
```

In [335]:

```
# Getting index

df12.index
```

Out[335]:

```
RangeIndex(start=0, stop=48, step=1)
```

Sorting and Ordering a DataFrame

In [336]:

```
df12
```

Out[336]:

	SalesRep	Region	Month	Sales	Units Sold
0	Amy	North	Jan	23040.0	239.0
1	Amy	North	Feb	24131.0	79.0
2	Amy	North	Mar	24646.0	71.0
3	Amy	North	Apr	22047.0	71.0
4	Amy	North	May	24971.0	NaN
5	Amy	North	Jun	24218.0	92.0
6	Amy	North	Jul	25735.0	175.0
7	Amy	North	Aug	NaN	87.0
8	Amy	North	Sep	25749.0	557.0
9	Amy	North	Oct	24437.0	95.0
10	Amy	North	Nov	25355.0	706.0
11	Amy	North	Dec	25899.0	180.0
12	Bob	North	Jan	20024.0	103.0
13	Bob	North	Feb	23822.0	267.0
14	Bob	North	Mar	24854.0	96.0
15	Bob	North	Apr	22838.0	74.0
16	Bob	North	May	25320.0	231.0
17	Bob	North	Jun	24733.0	164.0
18	Bob	North	Jul	21184.0	68.0
19	Bob	North	Aug	23174.0	114.0
20	Bob	North	Sep	25999.0	84.0
21	Bob	North	Oct	22639.0	260.0
22	Bob	North	Nov	23949.0	109.0
23	Bob	North	Dec	23179.0	465.0
24	Chuck	South	Jan	19886.0	95.0
25	Chuck	South	Feb	23494.0	148.0
26	Chuck	South	Mar	21824.0	83.0
27	Chuck	South	Apr	22058.0	96.0
28	Chuck	South	May	20280.0	453.0
29	Chuck	South	Jun	23965.0	760.0
30	Chuck	South	Jul	23032.0	155.0
31	Chuck	South	Aug	21273.0	769.0
32	Chuck	South	Sep	21584.0	114.0
33	Chuck	South	Oct	19625.0	83.0
34	Chuck	South	Nov	19832.0	70.0

	SalesRep	Region	Month	Sales	Units Sold
35	Chuck	South	Dec	20583.0	178.0
36	Doug	South	Jan	26264.0	92.0
37	Doug	South	Feb	29953.0	852.0
38	Doug	South	Mar	25041.0	86.0
39	Doug	South	Apr	29338.0	223.0
40	Doug	South	May	25150.0	242.0
41	Doug	South	Jun	27371.0	95.0
42	Doug	South	Jul	25044.0	82.0
43	Doug	South	Aug	29506.0	103.0
44	Doug	South	Sep	29061.0	146.0
45	Doug	South	Oct	27113.0	120.0
46	Doug	South	Nov	25953.0	81.0
47	Doug	South	Dec	28670.0	198.0

In [337]:

```
df12.sort_values(by = 'Sales')
```

Out[337]:

	SalesRep	Region	Month	Sales	Units Sold
33	Chuck	South	Oct	19625.0	83.0
34	Chuck	South	Nov	19832.0	70.0
24	Chuck	South	Jan	19886.0	95.0
12	Bob	North	Jan	20024.0	103.0
28	Chuck	South	May	20280.0	453.0
35	Chuck	South	Dec	20583.0	178.0
18	Bob	North	Jul	21184.0	68.0
31	Chuck	South	Aug	21273.0	769.0
32	Chuck	South	Sep	21584.0	114.0
26	Chuck	South	Mar	21824.0	83.0
3	Amy	North	Apr	22047.0	71.0
27	Chuck	South	Apr	22058.0	96.0
21	Bob	North	Oct	22639.0	260.0
15	Bob	North	Apr	22838.0	74.0
30	Chuck	South	Jul	23032.0	155.0
0	Amy	North	Jan	23040.0	239.0
19	Bob	North	Aug	23174.0	114.0
23	Bob	North	Dec	23179.0	465.0
25	Chuck	South	Feb	23494.0	148.0
13	Bob	North	Feb	23822.0	267.0
22	Bob	North	Nov	23949.0	109.0
29	Chuck	South	Jun	23965.0	760.0
1	Amy	North	Feb	24131.0	79.0
5	Amy	North	Jun	24218.0	92.0
9	Amy	North	Oct	24437.0	95.0
2	Amy	North	Mar	24646.0	71.0
17	Bob	North	Jun	24733.0	164.0
14	Bob	North	Mar	24854.0	96.0
4	Amy	North	May	24971.0	NaN
38	Doug	South	Mar	25041.0	86.0
42	Doug	South	Jul	25044.0	82.0
40	Doug	South	May	25150.0	242.0
16	Bob	North	May	25320.0	231.0
10	Amy	North	Nov	25355.0	706.0
6	Amy	North	Jul	25735.0	175.0

	SalesRep	Region	Month	Sales	Units Sold
8	Amy	North	Sep	25749.0	557.0
11	Amy	North	Dec	25899.0	180.0
46	Doug	South	Nov	25953.0	81.0
20	Bob	North	Sep	25999.0	84.0
36	Doug	South	Jan	26264.0	92.0
45	Doug	South	Oct	27113.0	120.0
41	Doug	South	Jun	27371.0	95.0
47	Doug	South	Dec	28670.0	198.0
44	Doug	South	Sep	29061.0	146.0
39	Doug	South	Apr	29338.0	223.0
43	Doug	South	Aug	29506.0	103.0
37	Doug	South	Feb	29953.0	852.0
7	Amy	North	Aug	NaN	87.0

Find Null Values or Check for Null Values

In [339]:

```
df12.isnull()
```

Out[339]:

	SalesRep	Region	Month	Sales	Units Sold
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	True
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	True	False
8	False	False	False	False	False
9	False	False	False	False	False
10	False	False	False	False	False
11	False	False	False	False	False
12	False	False	False	False	False
13	False	False	False	False	False
14	False	False	False	False	False
15	False	False	False	False	False
16	False	False	False	False	False
17	False	False	False	False	False
18	False	False	False	False	False
19	False	False	False	False	False
20	False	False	False	False	False
21	False	False	False	False	False
22	False	False	False	False	False
23	False	False	False	False	False
24	False	False	False	False	False
25	False	False	False	False	False
26	False	False	False	False	False
27	False	False	False	False	False
28	False	False	False	False	False
29	False	False	False	False	False
30	False	False	False	False	False
31	False	False	False	False	False
32	False	False	False	False	False
33	False	False	False	False	False
34	False	False	False	False	False

	SalesRep	Region	Month	Sales	Units Sold
35	False	False	False	False	False
36	False	False	False	False	False
37	False	False	False	False	False
38	False	False	False	False	False
39	False	False	False	False	False
40	False	False	False	False	False
41	False	False	False	False	False
42	False	False	False	False	False
43	False	False	False	False	False
44	False	False	False	False	False
45	False	False	False	False	False
46	False	False	False	False	False
47	False	False	False	False	False

In [340]:

```
df12.dropna()
```

Out[340]:

	SalesRep	Region	Month	Sales	Units Sold
0	Amy	North	Jan	23040.0	239.0
1	Amy	North	Feb	24131.0	79.0
2	Amy	North	Mar	24646.0	71.0
3	Amy	North	Apr	22047.0	71.0
5	Amy	North	Jun	24218.0	92.0
6	Amy	North	Jul	25735.0	175.0
8	Amy	North	Sep	25749.0	557.0
9	Amy	North	Oct	24437.0	95.0
10	Amy	North	Nov	25355.0	706.0
11	Amy	North	Dec	25899.0	180.0
12	Bob	North	Jan	20024.0	103.0
13	Bob	North	Feb	23822.0	267.0
14	Bob	North	Mar	24854.0	96.0
15	Bob	North	Apr	22838.0	74.0
16	Bob	North	May	25320.0	231.0
17	Bob	North	Jun	24733.0	164.0
18	Bob	North	Jul	21184.0	68.0
19	Bob	North	Aug	23174.0	114.0
20	Bob	North	Sep	25999.0	84.0
21	Bob	North	Oct	22639.0	260.0
22	Bob	North	Nov	23949.0	109.0
23	Bob	North	Dec	23179.0	465.0
24	Chuck	South	Jan	19886.0	95.0
25	Chuck	South	Feb	23494.0	148.0
26	Chuck	South	Mar	21824.0	83.0
27	Chuck	South	Apr	22058.0	96.0
28	Chuck	South	May	20280.0	453.0
29	Chuck	South	Jun	23965.0	760.0
30	Chuck	South	Jul	23032.0	155.0
31	Chuck	South	Aug	21273.0	769.0
32	Chuck	South	Sep	21584.0	114.0
33	Chuck	South	Oct	19625.0	83.0
34	Chuck	South	Nov	19832.0	70.0
35	Chuck	South	Dec	20583.0	178.0
36	Doug	South	Jan	26264.0	92.0

	SalesRep	Region	Month	Sales	Units Sold
37	Doug	South	Feb	29953.0	852.0
38	Doug	South	Mar	25041.0	86.0
39	Doug	South	Apr	29338.0	223.0
40	Doug	South	May	25150.0	242.0
41	Doug	South	Jun	27371.0	95.0
42	Doug	South	Jul	25044.0	82.0
43	Doug	South	Aug	29506.0	103.0
44	Doug	South	Sep	29061.0	146.0
45	Doug	South	Oct	27113.0	120.0
46	Doug	South	Nov	25953.0	81.0
47	Doug	South	Dec	28670.0	198.0

Replacing NaN values

In [341]:

```
df12.head(10)
```

Out[341]:

	SalesRep	Region	Month	Sales	Units Sold
0	Amy	North	Jan	23040.0	239.0
1	Amy	North	Feb	24131.0	79.0
2	Amy	North	Mar	24646.0	71.0
3	Amy	North	Apr	22047.0	71.0
4	Amy	North	May	24971.0	NaN
5	Amy	North	Jun	24218.0	92.0
6	Amy	North	Jul	25735.0	175.0
7	Amy	North	Aug	NaN	87.0
8	Amy	North	Sep	25749.0	557.0
9	Amy	North	Oct	24437.0	95.0

In [343]:

```
df12['Sales'] = df12['Sales'].fillna(df12['Sales'].mean())
```

In [344]:

```
df12.head(10)
```

Out[344]:

	SalesRep	Region	Month	Sales	Units Sold
0	Amy	North	Jan	23040.000000	239.0
1	Amy	North	Feb	24131.000000	79.0
2	Amy	North	Mar	24646.000000	71.0
3	Amy	North	Apr	22047.000000	71.0
4	Amy	North	May	24971.000000	NaN
5	Amy	North	Jun	24218.000000	92.0
6	Amy	North	Jul	25735.000000	175.0
7	Amy	North	Aug	24209.425532	87.0
8	Amy	North	Sep	25749.000000	557.0
9	Amy	North	Oct	24437.000000	95.0

Pivots

In [345]:

```
df12.pivot_table( values = 'Sales' , index = 'SalesRep' , columns = 'Month' )

# Aggregate function is 'mean' by default
```

Out[345]:

Month	Apr	Aug	Dec	Feb	Jan	Jul	Jun	Mar	May
SalesRep									
Amy	22047.0	24209.425532	25899.0	24131.0	23040.0	25735.0	24218.0	24646.0	24971.0
Bob	22838.0	23174.000000	23179.0	23822.0	20024.0	21184.0	24733.0	24854.0	25320.0
Chuck	22058.0	21273.000000	20583.0	23494.0	19886.0	23032.0	23965.0	21824.0	20280.0
Doug	29338.0	29506.000000	28670.0	29953.0	26264.0	25044.0	27371.0	25041.0	25150.0

In [346]:

```
df12.pivot_table( values = 'Sales' , index = 'SalesRep' , columns = 'Month', aggfunc = 'cou
```

Out[346]:

Month	Apr	Aug	Dec	Feb	Jan	Jul	Jun	Mar	May	Nov	Oct	Sep
SalesRep												
Amy	1	1	1	1	1	1	1	1	1	1	1	1
Bob	1	1	1	1	1	1	1	1	1	1	1	1
Chuck	1	1	1	1	1	1	1	1	1	1	1	1
Doug	1	1	1	1	1	1	1	1	1	1	1	1

Stats Exercises

In [2]:

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

In [3]:

```
import os
os.chdir('C:\\Users\\Swapnil bandekar\\Downloads\\Swapnil\\Data Analytics\\My Work\\Python\
os.getcwd()
```

Out[3]:

```
'C:\\Users\\Swapnil bandekar\\Downloads\\Swapnil\\Data Analytics\\My Work\\P
ython\\2. Introduction to Pandas'
```

Import the wind data

- daily average wind speeds for 1961-1978 at 12 synoptic meteorological stations in the Republic of Ireland (Haslett and raftery 1989).
- Each line corresponds to one day of data in the following format: year, month, day, average wind speed at each of the stations in the order given in Fig.4 of Haslett and Raftery : RPT, VAL, ROS, KIL, SHA, BIR, DUB, CLA, MUL, CLO, BEL, MAL

In [12]:

```
wind_data = pd.read_table('wind.data','\s+',parse_dates=[[0,1,2]])
wind_data.head()
```

Out[12]:

	Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
0	2061-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1	2061-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
2	2061-01-03	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
3	2061-01-04	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
4	2061-01-05	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83

Year 2061 ??? We know that data is from 1961 to 1978 , so we will clean the year data (fixing the century)

In [34]:

```
# We will define a function to fix the century and then will apply it to wind_data

def fix_century(x):
    year = x.year - 100 if x.year > 1978 else x.year
    return datetime.date( year , x.month , x.day )

wind_data['Yr_Mo_Dy'] = wind_data['Yr_Mo_Dy'].apply(fix_century)
wind_data.head()
```

Out[34]:

	Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
0	1961-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1	1961-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
2	1961-01-03	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
3	1961-01-04	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
4	1961-01-05	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83

In [35]:

```
wind_data.dtypes
```

Out[35]:

```
Yr_Mo_Dy     object
RPT         float64
VAL         float64
ROS         float64
KIL         float64
SHA         float64
BIR         float64
DUB         float64
CLA         float64
MUL         float64
CLO         float64
BEL         float64
MAL         float64
dtype: object
```

Changing the datatype of Date column to datetime and setting it as index

In [36]:

```
wind_data['Yr_Mo_Dy'] = pd.to_datetime(wind_data['Yr_Mo_Dy'])
wind_data.dtypes
```

Out[36]:

```
Yr_Mo_Dy    datetime64[ns]
RPT                float64
VAL                float64
ROS                float64
KIL                float64
SHA                float64
BIR                float64
DUB                float64
CLA                float64
MUL                float64
CLO                float64
BEL                float64
MAL                float64
dtype: object
```

In [37]:

```
wind_data.set_index('Yr_Mo_Dy',inplace=True)
wind_data.head()
```

Out[37]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
1961-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1961-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
1961-01-03	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
1961-01-04	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
1961-01-05	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83

Calculating the number of 'missing' values for each location

In [38]:

```
wind_data.isnull().sum()
```

Out[38]:

```
RPT    6
VAL    3
ROS    2
KIL    5
SHA    2
BIR    0
DUB    3
CLA    2
MUL    3
CLO    1
BEL    0
MAL    4
dtype: int64
```

Calculating the number of 'non-missing' values for each location

In [41]:

```
# Solution 1

wind_data.shape[0] - wind_data.isnull().sum()
```

Out[41]:

```
RPT    6568
VAL    6571
ROS    6572
KIL    6569
SHA    6572
BIR    6574
DUB    6571
CLA    6572
MUL    6571
CLO    6573
BEL    6574
MAL    6570
dtype: int64
```

In [42]:

```
# Solution 2

wind_data.notnull().sum()
```

Out[42]:

```
RPT    6568
VAL    6571
ROS    6572
KIL    6569
SHA    6572
BIR    6574
DUB    6571
CLA    6572
MUL    6571
CLO    6573
BEL    6574
MAL    6570
dtype: int64
```

Calculate the mean windspeeds of the windspeeds over all the locations and all the times

- A single number for the entire dataset.

In [57]:

```
wind_data.fillna(0).values.flatten().mean()

# values.flatten() : All the values are converted to 1-D Array
```

Out[57]:

```
10.223864592840483
```

Create a DataFrame called loc_stats and calculate the min, max and mean windspeeds and standard deviations of the windspeeds at each location over all the days

deviations of the windspeeds at each location over all the days

- A different set of numbers for each location

In [62]:

```
# Solution 1

loc_stats = wind_data.aggregate([min,max,'mean','std'])
loc_stats
```

Out[62]:

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA
min	0.670000	0.210000	1.500000	0.000000	0.130000	0.000000	0.000000	0.000000
max	35.800000	33.370000	33.840000	28.460000	37.540000	26.160000	30.370000	31.080000
mean	12.362987	10.644314	11.660526	6.306468	10.455834	7.092254	9.797343	8.495053
std	5.618413	5.267356	5.008450	3.605811	4.936125	3.968683	4.977555	4.499449

In [63]:

```
# Solution 2

loc_stats1 = wind_data.describe()
loc_stats1
```

Out[63]:

	RPT	VAL	ROS	KIL	SHA	BIR	I
count	6568.000000	6571.000000	6572.000000	6569.000000	6572.000000	6574.000000	6571.000
mean	12.362987	10.644314	11.660526	6.306468	10.455834	7.092254	9.797
std	5.618413	5.267356	5.008450	3.605811	4.936125	3.968683	4.977
min	0.670000	0.210000	1.500000	0.000000	0.130000	0.000000	0.000
25%	8.120000	6.670000	8.000000	3.580000	6.750000	4.000000	6.000
50%	11.710000	10.170000	10.920000	5.750000	9.960000	6.830000	9.210
75%	15.920000	14.040000	14.670000	8.420000	13.540000	9.670000	12.960
max	35.800000	33.370000	33.840000	28.460000	37.540000	26.160000	30.370

Create a DataFrame called day_stats and calculate the min, max and mean windspeed and standard deviations of the windspeeds across all the locations at each day.

- A different set of numbers for each day

In [67]:

```
# Solution 1

day_stats = wind_data.aggregate([min,max,'mean','std'],axis=1)
day_stats
```

Out[67]:

	min	max	mean	std
1961-01-01	9.29	18.50	13.018182	2.808875
1961-01-02	6.50	17.54	11.336364	3.188994
1961-01-03	6.17	18.50	11.641818	3.681912
1961-01-04	1.79	11.75	6.619167	3.198126
1961-01-05	6.17	13.33	10.630000	2.445356
...	...	...	...	...
1978-12-27	8.08	40.08	16.708333	7.868076
1978-12-28	5.00	41.46	15.150000	9.687857
1978-12-29	8.71	29.58	14.890000	5.756836
1978-12-30	9.13	28.79	15.367500	5.540437
1978-12-31	9.59	27.29	15.402500	5.702483

6574 rows × 4 columns

In [79]:

```
# Solution 2

day_stats1 = pd.DataFrame()

# We will calculate all the parameters one by one

day_stats1['min'] = wind_data.min(axis=1)
day_stats1['max'] = wind_data.max(axis=1)
day_stats1['mean'] = wind_data.mean(axis=1)
day_stats1['std_dev'] = wind_data.std(axis=1)
```

Find the average windspeed in January for each location

- Treat January 1961 and January 1962 both as January.

In [91]:

```
wind_data.loc[wind_data.index.month == 1].mean()
```

Out[91]:

```
RPT    14.847325
VAL    12.914560
ROS    13.299624
KIL     7.199498
SHA    11.667734
BIR     8.054839
DUB    11.819355
CLA     9.512047
MUL     9.543208
CLO    10.053566
BEL    14.550520
MAL    18.028763
dtype: float64
```

Downsample the record to a yearly frequency for each location

In [93]:

```
# Solution 1

wind_data.groupby(wind_data.index.year).mean()
```

Out[93]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CL
1961	12.299583	10.351796	11.362369	6.958227	10.881763	7.729726	9.733923	8.85878
1962	12.246923	10.110438	11.732712	6.960440	10.657918	7.393068	11.020712	8.79375
1963	12.813452	10.836986	12.541151	7.330055	11.724110	8.434712	11.075699	10.33654
1964	12.363661	10.920164	12.104372	6.787787	11.454481	7.570874	10.259153	9.46735
1965	12.451370	11.075534	11.848767	6.858466	11.024795	7.478110	10.618712	8.87991
1966	13.461973	11.557205	12.020630	7.345726	11.805041	7.793671	10.579808	8.83509
1967	12.737151	10.990986	11.739397	7.143425	11.630740	7.368164	10.652027	9.32561
1968	11.835628	10.468197	11.409754	6.477678	10.760765	6.067322	8.859180	8.25551
1969	11.166356	9.723699	10.902000	5.767973	9.873918	6.189973	8.564493	7.71139
1970	12.600329	10.726932	11.730247	6.217178	10.567370	7.609452	9.609890	8.33463
1971	11.273123	9.095178	11.088329	5.241507	9.440329	6.097151	8.385890	6.75731
1972	12.463962	10.561311	12.058333	5.929699	9.430410	6.358825	9.704508	7.68079
1973	11.828466	10.680493	10.680493	5.547863	9.640877	6.548740	8.482110	7.61427
1974	13.643096	11.811781	12.336356	6.427041	11.110986	6.809781	10.084603	9.89698
1975	12.008575	10.293836	11.564712	5.269096	9.190082	5.668521	8.562603	7.84383
1976	11.737842	10.203115	10.761230	5.109426	8.846339	6.311038	9.149126	7.14620
1977	13.099616	11.144493	12.627836	6.073945	10.003836	8.586438	11.523205	8.37838
1978	12.504356	11.044274	11.380000	6.082356	10.167233	7.650658	9.489342	8.80046

In [94]:

```
# Solution 2


wind_data.groupby(wind_data.index.to_period('A')).mean()
```

Out[94]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CL
1961	12.299583	10.351796	11.362369	6.958227	10.881763	7.729726	9.733923	8.85878
1962	12.246923	10.110438	11.732712	6.960440	10.657918	7.393068	11.020712	8.79375
1963	12.813452	10.836986	12.541151	7.330055	11.724110	8.434712	11.075699	10.33654
1964	12.363661	10.920164	12.104372	6.787787	11.454481	7.570874	10.259153	9.46735
1965	12.451370	11.075534	11.848767	6.858466	11.024795	7.478110	10.618712	8.87991
1966	13.461973	11.557205	12.020630	7.345726	11.805041	7.793671	10.579808	8.83509
1967	12.737151	10.990986	11.739397	7.143425	11.630740	7.368164	10.652027	9.32561
1968	11.835628	10.468197	11.409754	6.477678	10.760765	6.067322	8.859180	8.25551
1969	11.166356	9.723699	10.902000	5.767973	9.873918	6.189973	8.564493	7.71139
1970	12.600329	10.726932	11.730247	6.217178	10.567370	7.609452	9.609890	8.33463
1971	11.273123	9.095178	11.088329	5.241507	9.440329	6.097151	8.385890	6.75731
1972	12.463962	10.561311	12.058333	5.929699	9.430410	6.358825	9.704508	7.68079
1973	11.828466	10.680493	10.680493	5.547863	9.640877	6.548740	8.482110	7.61427
1974	13.643096	11.811781	12.336356	6.427041	11.110986	6.809781	10.084603	9.89698
1975	12.008575	10.293836	11.564712	5.269096	9.190082	5.668521	8.562603	7.84383
1976	11.737842	10.203115	10.761230	5.109426	8.846339	6.311038	9.149126	7.14620
1977	13.099616	11.144493	12.627836	6.073945	10.003836	8.586438	11.523205	8.37838
1978	12.504356	11.044274	11.380000	6.082356	10.167233	7.650658	9.489342	8.80046

Downsample the record to a monthly frequency for each location

In [100]:

```
wind_data.groupby(wind_data.index.to_period('M')).mean()
```

Out[100]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CI
1961-01	14.841333	11.988333	13.431613	7.736774	11.072759	8.588065	11.184839	9.2453
1961-02	16.269286	14.975357	14.441481	9.230741	13.852143	10.937500	11.890714	11.8460
1961-03	10.890000	11.296452	10.752903	7.284000	10.509355	8.866774	9.644194	9.8296
1961-04	10.722667	9.427667	9.998000	5.830667	8.435000	6.495000	6.925333	7.0946
1961-05	9.860968	8.850000	10.818065	5.905333	9.490323	6.574839	7.604000	8.1770
...	...	...	...	...	...	...	...	
1978-08	9.645161	8.259355	9.032258	4.502903	7.368065	5.935161	5.650323	5.4177
1978-09	10.913667	10.895000	10.635000	5.725000	10.372000	9.278333	10.790333	9.5830
1978-10	9.897742	8.670968	9.295806	4.721290	8.525161	6.774194	8.115484	7.3377
1978-11	16.151667	14.802667	13.508000	7.317333	11.475000	8.743000	11.492333	9.6573
1978-12	16.175484	13.748065	15.635161	7.094839	11.398710	9.241613	12.077419	10.1948

216 rows × 12 columns

Downsample the record to a weekly frequency for each location

In [101]:

```
wind_data.groupby(wind_data.index.to_period('W')).mean()
```

Out[101]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	C
1960-12-26/1961-01-01	15.040000	14.960000	13.170000	9.290000	NaN	9.870000	13.670000	10.250
1961-01-02/1961-01-08	13.541429	11.486667	10.487143	6.417143	9.474286	6.435714	11.061429	6.616
1961-01-09/1961-01-15	12.468571	8.967143	11.958571	4.630000	7.351429	5.072857	7.535714	6.820
1961-01-16/1961-01-22	13.204286	9.862857	12.982857	6.328571	8.966667	7.417143	9.257143	7.875
1961-01-23/1961-01-29	19.880000	16.141429	18.225714	12.720000	17.432857	14.828571	15.528571	15.160
...	...	...	...	...	...	...	...	
1978-11-27/1978-12-03	14.934286	11.232857	13.941429	5.565714	10.215714	8.618571	9.642857	7.685
1978-12-04/1978-12-10	20.740000	19.190000	17.034286	9.777143	15.287143	12.774286	14.437143	12.488
1978-12-11/1978-12-17	16.758571	14.692857	14.987143	6.917143	11.397143	7.272857	10.208571	7.967
1978-12-18/1978-12-24	11.155714	8.008571	13.172857	4.004286	7.825714	6.290000	7.798571	8.667
1978-12-25/1978-12-31	14.951429	11.801429	16.035714	6.507143	9.660000	8.620000	13.708571	10.477

940 rows × 12 columns

Calculate the min, max and mean windspeeds and standard deviations of the windspeeds across all locations for each week (assume that the first week starts on January 2 1961) for the first 52 weeks

In [121]:

```
# Solution 1

weekly_stats = wind_data.groupby(wind_data.index.to_period('W')).aggregate([min,max,'mean',
weekly_stats.iloc[1:53]
```

Out[121]:

	RPT				VAL				ROS		...	
	min	max	mean	std	min	max	mean	std	min	max	...	mean
Yr_Mo_Dy												
1961-01-02/1961-01-08	10.58	18.50	13.541429	2.631321	6.63	16.88	11.486667	3.949525	7.62	12.33	...	8.497143
1961-01-09/1961-01-15	9.04	19.75	12.468571	3.555392	3.54	12.08	8.967143	3.148945	7.08	19.50	...	7.571429
1961-01-16/1961-01-22	4.92	19.83	13.204286	5.337402	3.42	14.37	9.862857	3.837785	7.29	20.79	...	8.124286
1961-01-23/1961-01-29	13.62	25.04	19.880000	4.619061	9.96	23.91	16.141429	5.170224	12.67	25.84	...	15.640000

In [124]:

```
# Solution 2

weekly_stats1 = wind_data.resample('W').aggregate([min,max,'mean','std'])
weekly_stats1.iloc[1:53]
```

Out[124]:

	RPT				VAL				ROS		...	
	min	max	mean	std	min	max	mean	std	min	max	...	mean
Yr_Mo_Dy												
1961-01-08	10.58	18.50	13.541429	2.631321	6.63	16.88	11.486667	3.949525	7.62	12.33	...	8.497143
1961-01-15	9.04	19.75	12.468571	3.555392	3.54	12.08	8.967143	3.148945	7.08	19.50	...	7.571429
1961-01-22	4.92	19.83	13.204286	5.337402	3.42	14.37	9.862857	3.837785	7.29	20.79	...	8.124286
1961-01-29	13.62	25.04	19.880000	4.619061	9.96	23.91	16.141429	5.170224	12.67	25.84	...	15.640000
1961-02-05	10.58	24.21	16.827143	5.251408	9.46	24.21	15.460000	5.187395	9.04	19.70	...	9.460000

In [126]:

```
# Alternate way for indexing

weekly_stats1.loc[weekly_stats1.index[1:53]]
```

Out[126]:

	RPT				VAL				ROS		...	
	min	max	mean	std	min	max	mean	std	min	max	...	mean
Yr_Mo_Dy												
1961-01-08	10.58	18.50	13.541429	2.631321	6.63	16.88	11.486667	3.949525	7.62	12.33	...	8.497143
1961-01-15	9.04	19.75	12.468571	3.555392	3.54	12.08	8.967143	3.148945	7.08	19.50	...	7.571429
1961-01-22	4.92	19.83	13.204286	5.337402	3.42	14.37	9.862857	3.837785	7.29	20.79	...	8.124286
1961-01-29	13.62	25.04	19.880000	4.619061	9.96	23.91	16.141429	5.170224	12.67	25.84	...	15.640000
1961-02-05	10.58	24.21	16.827143	5.251408	9.46	24.21	15.460000	5.187395	9.04	19.70	...	9.460000