In [10]:

```
print(type(as_json))
```

```
<class 'str'>
```

In [ ]:

```

```

In [13]:

```
result
```

Out[13]:

```
{'name': 'Steve',
 'places_lived': ['India', 'USA', 'Spain', 'Germany'],
 'pet': None,
 'sibling': [{'name': 'Henry', 'age': 30, 'pets': ['Alpha', 'Beta']},
  {'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Bru', 'Cisco']}]}
```

In [14]:

```
result['sibling']
```

Out[14]:

```
[{'name': 'Henry', 'age': 30, 'pets': ['Alpha', 'Beta']},
 {'name': 'Katie', 'age': 38, 'pets': ['Sixes', 'Bru', 'Cisco']}]
```

In [12]:

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

In [15]:

```
pd.DataFrame(result['sibling'],columns=['name','age'])
```

Out[15]:

	name	age
0	Henry	30
1	Katie	38

In [ ]:

```

```

In [16]:

```
data = pd.read_json('data.json')
```

In [17]:

```
data
```

Out[17]:

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

In [ ]:

```
pd.read_excel('demo.xlsx','Rate Card')
```

In [ ]:

```

```

Data Cleansing and Prerperation

In [ ]:

```

```

Handling Missing Data

In [18]:

```
string_data = pd.Series(['aardvark','alpha',np.nan,'avacado'])
```

In [19]:

```
print(string_data)
```

```
0    aardvark
1       alpha
2         NaN
3     avacado
dtype: object
```

In [20]:

```
string_data.isnull()  #this is used to check which records are null and the result is thrown in boolean
```

Out[20]:

```
0    False
1    False
2     True
3    False
dtype: bool
```

In [ ]:

```

```

filter missing data

In [21]:

```
ser1 = pd.Series([1,np.nan,3.5,np.nan])
```

In [22]:

```
print(ser1)
```

```
0    1.0
1    NaN
2    3.5
3    NaN
dtype: float64
```

In [23]:

```
ser1.dropna()
```

Out[23]:

```
0    1.0
2    3.5
dtype: float64
```

In [25]:

```
ser1[ser1.notnull()]
```

Out[25]:

```
0    1.0
2    3.5
dtype: float64
```

In [ ]:

```

```

In [ ]:

```

```

In [26]:

```
data = pd.DataFrame([[1,6.5,3],[1,np.nan,np.nan],[1,np.nan,np.nan],[np.nan,np.nan,np.nan],[np.nan,6.5,3]])
```

In [27]:

```
1 data
```

Out[27]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	1.0	NaN	NaN
3	NaN	NaN	NaN
4	NaN	6.5	3.0

In [28]:

```
1 cleaned_data = data.dropna()
```

In [29]:

```
1 cleaned_data
```

Out[29]:

	0	1	2
0	1.0	6.5	3.0

In [30]:

```
1 data.dropna(how='all')
```

Out[30]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	1.0	NaN	NaN
4	NaN	6.5	3.0

In [31]:

```
1 data.dropna(axis=1,how='all')
```

Out[31]:

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	1.0	NaN	NaN
3	NaN	NaN	NaN
4	NaN	6.5	3.0

In [ ]:

```
1
```

In [ ]:

```
1
```

In [53]:

```
1 df = pd.DataFrame(np.random.randn(7,3))
```

In [33]:

```
1 df
```

Out[33]:

	0	1	2
0	-1.177288	-0.401541	-0.446020
1	-1.519291	-0.683789	1.412587
2	0.275131	-0.878904	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [55]:

```
1 df.iloc[0:3,1] = np.nan
```

In [36]:

```
1 df
```

Out[36]:

	0	1	2
0	-1.177288	NaN	-0.446020
1	-1.519291	NaN	1.412587
2	0.275131	NaN	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [56]:

```
1 df.iloc[0:2,2] = np.nan
```

In [38]:

```
1 df
```

Out[38]:

	0	1	2
0	-1.177288	NaN	NaN
1	-1.519291	NaN	NaN
2	0.275131	NaN	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [ ]:

```
1
```

handling missing data

In [39]:

```
1 df
```

Out[39]:

	0	1	2
0	-1.177288	NaN	NaN
1	-1.519291	NaN	NaN
2	0.275131	NaN	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [41]:

```
1 df.fillna(0) #throwing copy
```

Out[41]:

	0	1	2
0	-1.177288	0.000000	0.000000
1	-1.519291	0.000000	0.000000
2	0.275131	0.000000	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [42]:

```
1 df
```

Out[42]:

	0	1	2
0	-1.177288	NaN	NaN
1	-1.519291	NaN	NaN
2	0.275131	NaN	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [50]:

```
1 df
```

Out[50]:

	0	1	2
0	-1.177288	0.000000	0.000000
1	-1.519291	0.000000	0.000000
2	0.275131	0.000000	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [43]:

```
df.fillna({1:0.5,2:0})
```

Out[43]:

	0	1	2
0	-1.177288	0.500000	0.000000
1	-1.519291	0.500000	0.000000
2	0.275131	0.500000	-0.724181
3	-0.475731	1.473696	0.819303
4	2.115289	-0.968340	-1.475501
5	-0.680491	-0.701993	-2.298830
6	0.055987	-0.909124	0.115106

In [51]:

```
df.fillna(0,inplace=True)
```

In [57]:

```
df
```

Out[57]:

	0	1	2
0	-0.696944	NaN	NaN
1	0.991113	NaN	NaN
2	0.069740	NaN	-0.747242
3	-2.183064	2.158640	-0.320176
4	0.824128	0.799570	-1.181073
5	0.383022	-0.445819	0.100012
6	0.219232	0.295120	0.800224

In [58]:

```
df1 = df.fillna(0)
```

In [59]:

```
df1
```

Out[59]:

	0	1	2
0	-0.696944	0.000000	0.000000
1	0.991113	0.000000	0.000000
2	0.069740	0.000000	-0.747242
3	-2.183064	2.158640	-0.320176
4	0.824128	0.799570	-1.181073
5	0.383022	-0.445819	0.100012
6	0.219232	0.295120	0.800224

In [60]:

```
df
```

Out[60]:

	0	1	2
0	-0.696944	NaN	NaN
1	0.991113	NaN	NaN
2	0.069740	NaN	-0.747242
3	-2.183064	2.158640	-0.320176
4	0.824128	0.799570	-1.181073
5	0.383022	-0.445819	0.100012
6	0.219232	0.295120	0.800224

In [ ]:

```

```

In [61]:

```
df.fillna(0,inplace=True)
```

In [62]:

```
df
```

Out[62]:

	0	1	2
0	-0.696944	0.000000	0.000000
1	0.991113	0.000000	0.000000
2	0.069740	0.000000	-0.747242
3	-2.183064	2.158640	-0.320176
4	0.824128	0.799570	-1.181073
5	0.383022	-0.445819	0.100012
6	0.219232	0.295120	0.800224

In [ ]:

```

```

In [ ]:

```

```

In [63]:

```
df = pd.DataFrame(np.random.randn(6,3))
```

In [64]:

```
df
```

Out[64]:

	0	1	2
0	-0.991670	-0.523700	-0.282011
1	0.151081	0.005041	0.117496
2	0.041059	1.388625	-0.199624
3	-0.529330	1.140412	-0.442659
4	-1.825217	-3.082615	1.446838
5	0.982442	-1.147429	-0.643398

In [65]:

```
df.iloc[2:,1] = np.nan
df.iloc[4:,2] = np.nan
```

In [66]:

```
df
```

Out[66]:

	0	1	2
0	-0.991670	-0.523700	-0.282011
1	0.151081	0.005041	0.117496
2	0.041059	NaN	-0.199624
3	-0.529330	NaN	-0.442659
4	-1.825217	NaN	NaN
5	0.982442	NaN	NaN

In [67]:

```
df.fillna(method='ffill')
```

Out[67]:

	0	1	2
0	-0.991670	-0.523700	-0.282011
1	0.151081	0.005041	0.117496
2	0.041059	0.005041	-0.199624
3	-0.529330	0.005041	-0.442659
4	-1.825217	0.005041	-0.442659
5	0.982442	0.005041	-0.442659

In [68]:

```
df.fillna(method='ffill',limit=2)
```

Out[68]:

	0	1	2
0	-0.991670	-0.523700	-0.282011
1	0.151081	0.005041	0.117496
2	0.041059	0.005041	-0.199624
3	-0.529330	0.005041	-0.442659
4	-1.825217	NaN	-0.442659
5	0.982442	NaN	-0.442659

In [ ]:

```

```

In [70]:

```
data = pd.Series([1,np.nan,3.5,np.nan])
```

In [71]:

```
data
```

Out[71]:

```
0    1.0
1    NaN
2    3.5
3    NaN
dtype: float64
```

In [72]:

```
data.fillna(data.mean())
```

Out[72]:

```
0    1.00
1    2.25
2    3.50
3    2.25
dtype: float64
```

In [ ]:

```

```

Removing Duplicates

In [73]:

```
data = pd.DataFrame(
    {'k1':['one','two']*3 + ['two'],
    'k2':[1,1,2,3,3,4,4]
    })
```

In [74]:

```
1 data
```

Out[74]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

In [75]:

```
1 data.duplicated()
```

Out[75]:

```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

In [76]:

```
1 data.drop_duplicates()
```

Out[76]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

In [ ]:

```
1
```

In [77]:

```
1 data['v1'] = range(7)
```

In [78]:

```
1 data
```

Out[78]:

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5
6	two	4	6

In [79]:

```
1 data.drop_duplicates(['k1']) #dropping duplicate on basis of col
```

Out[79]:

	k1	k2	v1
0	one	1	0
1	two	1	1

In [ ]:

```
1
```

In [80]:

```
1 data
```

Out[80]:

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5
6	two	4	6

In [83]:

```
1 data.drop_duplicates(['k1','k2'])
```

Out[83]:

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5

In [86]:

```
1 data['k1'][3] = 'TEN'
```

```
<ipython-input-86-28d67488334f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-vi
ew-versus-a-copy)
  data['k1'][3] = 'TEN'
```

In [87]:

```
1 data['k1']
```

Out[87]:

```
0    one
1    two
2    one
3    TEN
4    one
5    two
6    two
Name: k1, dtype: object
```

In [ ]:

```
1
```

Transforming Data Using Function or Mapping

In [89]:

```
1 data = pd.DataFrame({
2     "food":[
3         'apple','egg','wheat','bread','rice','dal','grapes','pineapple','honey'
4     ],
5     'calories':[4,3,12,6,7.5,8,3,5,6]})
```

In [90]:

```
data
```

Out[90]:

	food	calories
0	apple	4.0
1	egg	3.0
2	wheat	12.0
3	bread	6.0
4	rice	7.5
5	dal	8.0
6	grapes	3.0
7	pineapple	5.0
8	honey	6.0

In [91]:

```
food_categories = {
    'apple':'fruit',
    'egg':'non veg food',
    'wheat':'grain',
    'bread':'veg food',
    'rice':'grain',
    'dal':'grain',
    'grapes':'fruit',
    'pineapple':'fruit',
    'honey':'veg food'
}
```

In [94]:

```
data['food'] = data['food'].str.upper()
```

In [95]:

```
data
```

Out[95]:

	food	calories
0	APPLE	4.0
1	EGG	3.0
2	WHEAT	12.0
3	BREAD	6.0
4	RICE	7.5
5	DAL	8.0
6	GRAPES	3.0
7	PINEAPPLE	5.0
8	HONEY	6.0

In [96]:

```
data['food'] = data['food'].str.lower()
```

In [98]:

```
data['food']
```

Out[98]:

```
0        apple
1          egg
2        wheat
3        bread
4         rice
5          dal
6       grapes
7    pineapple
8        honey
Name: food, dtype: object
```

In [99]:

```
data['FoodCategory'] = data['food'].map(food_categories)
```

In [100]:

```
1 data
```

Out[100]:

	food	calories	FoodCategory
0	apple	4.0	fruit
1	egg	3.0	non veg food
2	wheat	12.0	grain
3	bread	6.0	veg food
4	rice	7.5	grain
5	dal	8.0	grain
6	grapes	3.0	fruit
7	pineapple	5.0	fruit
8	honey	6.0	veg food

In [102]:

```
1 food_categories
```

Out[102]:

```
{'apple': 'fruit',
 'egg': 'non veg food',
 'wheat': 'grain',
 'bread': 'veg food',
 'rice': 'grain',
 'dal': 'grain',
 'grapes': 'fruit',
 'pineapple': 'fruit',
 'honey': 'veg food'}
```

In [ ]:

```
1
```

In [103]:

```
1 data = pd.Series([1,-999,2,-999,-1000,3])
```

In [104]:

```
1 data
```

Out[104]:

```
0       1
1    -999
2       2
3    -999
4   -1000
5       3
dtype: int64
```

In [105]:

```
1 data.replace(-999,np.nan)
```

Out[105]:

```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

In [106]:

```
1 data.replace([-999,-1000],np.nan)
```

Out[106]:

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

In [107]:

```
data.replace([-999,-1000],[np.nan,0])
```

Out[107]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [108]:

```
data.replace({-999:np.nan,-1000:0})
```

Out[108]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [ ]:

```

```

Renaming Axis Indexes

In [109]:

```
data = pd.DataFrame(np.arange(12).reshape(3,4),
                    index=['Mumbai','Pune','Chennai'],
                  columns=['one','two','three','four'])
```

In [110]:

```
data
```

Out[110]:

	one	two	three	four
Mumbai	0	1	2	3
Pune	4	5	6	7
Chennai	8	9	10	11

In [ ]:

```

```

In [113]:

```
to_upper = lambda x: x.upper()
```

In [116]:

```
data.index.map(to_upper)
```

Out[116]:

```
Index(['MUMBAI', 'PUNE', 'CHENNAI'], dtype='object')
```

In [115]:

```
data.index
```

Out[115]:

```
Index(['Mumbai', 'Pune', 'Chennai'], dtype='object')
```

In [ ]:

```

```

In [117]:

```
1 data.rename(columns=str.upper)
```

Out[117]:

	ONE	TWO	THREE	FOUR
Mumbai	0	1	2	3
Pune	4	5	6	7
Chennai	8	9	10	11

In [ ]:

```
1
```

Binning

In [118]:

```
1 ages = [20,22,25,27,21,23,37,31,61,45,41,32]
```

In [119]:

```
1 bins = [18,25,35,60,100]
```

In [120]:

```
1 category = pd.cut(ages,bins)
```

In [121]:

```
1 category
```

Out[121]:

```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

In [122]:

```
1 pd.value_counts(category)
```

Out[122]:

```
(18, 25]     5
(25, 35]     3
(35, 60]     3
(60, 100]    1
dtype: int64
```

In [ ]:

```
1
```