In [110]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
from numpy import nan as NA
```

Handling Missing Data

In [2]:
```
string_data = pd.Series(["Karachi" , "Hyderabad" , np.nan ,"Lahore"])
string_data
```

Out[2]:
```
0      Karachi
1    Hyderabad
2          NaN
3       Lahore
dtype: object
```

In [3]:
```
string_data.isnull()
```

Out[3]:
```
0    False
1    False
2     True
3    False
dtype: bool
```

In [4]:
```
string_data[0] = None
```

In [5]:
```
string_data
```

Out[5]:
```
0         None
1    Hyderabad
2          NaN
3       Lahore
dtype: object
```

In [6]:
```
string_data.isnull()
```

Out[6]:
```
0     True
1    False
2     True
3    False
dtype: bool
```

Filtering Out Missing Data

In [8]:
```
data = pd.Series([1 , NA , 3.5 , NA , 7])
data
```

Out[8]:
```
0    1.0
1    NaN
2    3.5
3    NaN
4    7.0
dtype: float64
```

In [9]:
```
data.dropna()
```

Out[9]:
```
0    1.0
2    3.5
4    7.0
dtype: float64
```

In [14]:
```
data = pd.DataFrame([[1,3,7],[NA,NA,NA],[5,8,9.5],[5,9.5,NA]])
data
```

Out[14]:

	0	1	2
0	1.0	3.0	7.0
1	NaN	NaN	NaN
2	5.0	8.0	9.5
3	5.0	9.5	NaN

In [16]:
```
cleaned = data.dropna()
cleaned
```

Out[16]:

	0	1	2
0	1.0	3.0	7.0
2	5.0	8.0	9.5

Passing how='all' will only drop rows that are all contain only NA

In [19]:
```
data.dropna(how = 'all')
```

Out[19]:

	0	1	2
0	1.0	3.0	7.0
2	5.0	8.0	9.5
3	5.0	9.5	NaN

Droping column we need to give axis = 1 with how = "all"

In [21]:
```
data[2] = NA
```

In [24]:
```
data.dropna(axis = 1 , how = 'all')
```

Out[24]:

	0	1
0	1.0	3.0
1	NaN	NaN
2	5.0	8.0
3	5.0	9.5

In [25]:
```
df = pd.DataFrame(np.random.randn(7,3))
df
```

Out[25]:

	0	1	2
0	1.100849	-2.272636	0.176591
1	0.145582	0.390221	-0.172730
2	0.590904	-0.296096	-0.172002
3	-2.319067	1.118964	-0.052525
4	-1.160303	-1.280268	-0.461312
5	-0.560320	0.616657	0.004186
6	1.253843	0.933968	-0.642544

In [26]:
```
df.iloc[:4,1] = NA
```

In [28]:
```
df.iloc[:2,2] = NA
```

In [29]:
```
df
```

Out[29]:

	0	1	2
0	1.100849	NaN	NaN
1	0.145582	NaN	NaN
2	0.590904	NaN	-0.172002
3	-2.319067	NaN	-0.052525
4	-1.160303	-1.280268	-0.461312
5	-0.560320	0.616657	0.004186
6	1.253843	0.933968	-0.642544

In [30]:
```
df.dropna()
```

Out[30]:

	0	1	2
4	-1.160303	-1.280268	-0.461312
5	-0.560320	0.616657	0.004186
6	1.253843	0.933968	-0.642544

In [37]:
```

```

Out[37]:

	0	1	2
0	1.100849	NaN	NaN
1	0.145582	NaN	NaN
2	0.590904	NaN	-0.172002
3	-2.319067	NaN	-0.052525
4	-1.160303	-1.280268	-0.461312
5	-0.560320	0.616657	0.004186
6	1.253843	0.933968	-0.642544

Filling missing values

In [39]:
```
df.fillna(0)
```

Out[39]:

	0	1	2
0	1.100849	0.000000	0.000000
1	0.145582	0.000000	0.000000
2	0.590904	0.000000	-0.172002
3	-2.319067	0.000000	-0.052525
4	-1.160303	-1.280268	-0.461312
5	-0.560320	0.616657	0.004186
6	1.253843	0.933968	-0.642544

In [40]:
```
df.fillna({1:0.5,2:0.75})
```

Out[40]:

	0	1	2
0	1.100849	0.500000	0.750000
1	0.145582	0.500000	0.750000
2	0.590904	0.500000	-0.172002
3	-2.319067	0.500000	-0.052525
4	-1.160303	-1.280268	-0.461312
5	-0.560320	0.616657	0.004186
6	1.253843	0.933968	-0.642544

Interpolation Method with reindexing

In [41]:
```
dataset = pd.DataFrame(np.random.rand(6,3))
dataset
```

Out[41]:

	0	1	2
0	0.330678	0.672752	0.121038
1	0.754963	0.356933	0.873204
2	0.767359	0.261480	0.904346
3	0.717064	0.783731	0.496224
4	0.890216	0.890996	0.148290
5	0.019688	0.994067	0.177524

In [42]:
```
dataset.iloc[2:,1] = NA
dataset.iloc[4:,2] = NA
```

In [43]:
```
dataset
```

Out[43]:

	0	1	2
0	0.330678	0.672752	0.121038
1	0.754963	0.356933	0.873204
2	0.767359	NaN	0.904346
3	0.717064	NaN	0.496224
4	0.890216	NaN	NaN
5	0.019688	NaN	NaN

'ffill' stands for 'forward fill' and will propagate last valid observation forward.

In [45]:
```
dataset.fillna(method='ffill')
```

Out[45]:

	0	1	2
0	0.330678	0.672752	0.121038
1	0.754963	0.356933	0.873204
2	0.767359	0.356933	0.904346
3	0.717064	0.356933	0.496224
4	0.890216	0.356933	0.496224
5	0.019688	0.356933	0.496224

In [48]:
```
dataset.fillna(method='ffill' , limit = 2)
```

Out[48]:

	0	1	2
0	0.330678	0.672752	0.121038
1	0.754963	0.356933	0.873204
2	0.767359	0.356933	0.904346
3	0.717064	0.356933	0.496224
4	0.890216	NaN	0.496224
5	0.019688	NaN	0.496224

Data Transformation

Removing Duplicates

In [59]:
```
data_set = pd.DataFrame({'k1':['one','two'] * 3 + ['two']})
data_set
```

Out[59]:

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

In [60]:
```
data_set.duplicated(keep = 'first')
```

Out[60]:
```
0    False
1    False
2     True
3     True
4     True
5     True
6     True
dtype: bool
```

In [61]:
```
data_set.drop_duplicates(subset = None , keep = 'first')
```

Out[61]:

	k1
0	one
1	two

Transforming Data Using a Function or Mapping

In [63]:
```
data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
 'Pastrami', 'corned beef', 'Bacon',
 'pastrami', 'honey ham', 'nova lox'],
 'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
```

In [64]:
```
data
```

Out[64]:

	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

In [65]:
```
meat_to_animal = {
 'bacon': 'pig',
 'pulled pork': 'pig',
 'pastrami': 'cow',
 'corned beef': 'cow',
 'honey ham': 'pig',
 'nova lox': 'salmon'
}
```

In [66]:
```
meat_to_animal
```

Out[66]:
```
{'bacon': 'pig',
 'pulled pork': 'pig',
 'pastrami': 'cow',
 'corned beef': 'cow',
 'honey ham': 'pig',
 'nova lox': 'salmon'}
```

Before merging we need to lowercase some food items

In [71]:
```
lowercased = data['food'].str.lower()
```

In [72]:
```
data['animal'] = lowercased.map(meat_to_animal)
```

In [73]:
```
data
```

Out[73]:

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	Pastrami	6.0	cow
4	corned beef	7.5	cow
5	Bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

Replacing Values

In [74]:
```
data_rep = pd.Series([1., -999., 2., -999., -1000., 3.])
```

In [75]:
```
data_rep
```

Out[75]:
```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

In [76]:
```
data_rep.replace(-999,np.nan)
```

Out[76]:
```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

In [80]:
```
data_rep.replace([-999,-1000],[np.nan,0])
```

Out[80]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

Detecting and Filtering Outliers

In [162]:
```
dataset_outlier = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1000]
```

In [163]:
```
dataset_outlier
```

Out[163]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1000]
```

Detecting outlier through Z-Score

Formula for Z score = (Observation - Mean) / Standard Deviation

In [164]:
```
outliers = []
def detect_outliers(data):

    threshold = 3
    mean = np.mean(data)
    std = np.std(data)


    for i in data:
        z_score = (i-mean)/std
        if np.abs(z_score) > threshold:
            outliers.append(i)
    return outliers
```

In [165]:
```
r = detect_outliers(dataset_outlier)
r
```

Out[165]:
```
[1000]
```

In [ ]:
```

```