Preprocessing

In [42]:
```python
import pandas as pd
import numpy as np
from sklearn import preprocessing
```

Load Data

In [2]:
```python
country = pd.read_csv('c://c_data.csv', encoding='ansi', header=2)
```

In [3]:
```
country = country.rename(columns={'Country Name':'Name', 'Country Code':'Code',
                                  'Population growth':'pop_growth', 'Total population':'p
op',
                                   'Area (sq. km)':'Area'})
country
```

Out[3]:

	Name	Code	pop_growth	pop	Area
0	Brazil	BRA	0.817556	2.076529e+08	8358140.0
1	Switzerland	CHE	1.077221	8.372098e+06	39516.0
2	Germany	DEU	1.193867	8.266768e+07	348900.0
3	Denmark	DNK	0.834638	NaN	42262.0
4	Spain	ESP	-0.008048	4.644396e+07	500210.0
5	France	FRA	0.407491	6.689611e+07	547557.0
6	Japan	JPN	-0.115284	1.269945e+08	364560.0
7	Greece	GRC	-0.687543	1.074674e+07	128900.0
8	Iran	IRN	1.148789	8.027743e+07	1628760.0
9	Kuwait	KWT	2.924206	4.052584e+06	NaN
10	Morocco	MAR	NaN	3.527679e+07	446300.0
11	Nigeria	NGA	2.619034	1.859896e+08	910770.0
12	Qatar	QAT	3.495070	2.569804e+06	11610.0
13	Sweden	SWE	NaN	9.903122e+06	407310.0
14	India	IND	1.148215	1.324171e+09	2973190.0
15	World	WLD	1.181680	7.442136e+09	129733172.7

In [4]:
```
country[country.Name=='Iran']
```

Out[4]:

	Name	Code	pop_growth	pop	Area
8	Iran	IRN	1.148789	80277428.0	1628760.0

In [5]:
```
country.shape
```

Out[5]: (16, 5)

In [6]:
```
country.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 5 columns):
Name          16 non-null object
Code          16 non-null object
pop_growth    14 non-null float64
pop           15 non-null float64
Area          15 non-null float64
dtypes: float64(3), object(2)
memory usage: 720.0+ bytes
```

The .info() method provides important information about a DataFrame

In [7]:
```
country.describe()
```

Out[7]:

	pop_growth	pop	Area
count	14.000000	1.500000e+01	1.500000e+01
mean	1.145492	6.422767e+08	9.762744e+06
std	1.173195	1.909868e+09	3.325701e+07
min	-0.687543	2.569804e+06	1.161000e+04
25%	0.510007	1.032493e+07	2.389000e+05
50%	1.112718	6.689611e+07	4.463000e+05
75%	1.190820	1.564921e+08	1.269765e+06
max	3.495070	7.442136e+09	1.297332e+08

In [8]:
```
max_pop = country['pop'].max()
country['pop'][country['pop']==max_pop]
```

Out[8]:
```
15    7.442136e+09
Name: pop, dtype: float64
```

In [ ]:
```
country.drop('World', axis=0, inplace=True)
```

Missing Value

> NaN

In [ ]:
```
country = country.fillna('?')
```

In [ ]:
```
country.head()
```

Count NaN in a DataFrame

In [ ]:
```
country.isnull()
```

In [10]:
```
country.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 5 columns):
Name          16 non-null object
Code          16 non-null object
pop_growth    14 non-null float64
pop           15 non-null float64
Area          15 non-null float64
dtypes: float64(3), object(2)
memory usage: 720.0+ bytes
```

In [11]:
```
country.replace('?', np.nan, inplace=True)
```

In [12]:
```
country.isnull().sum()
```

Out[12]:
```
Name          0
Code          0
pop_growth    2
pop           1
Area          1
dtype: int64
```

In [13]:
```
country.dropna(axis=0)
```

Out[13]:

	Name	Code	pop_growth	pop	Area
0	Brazil	BRA	0.817556	2.076529e+08	8358140.0
1	Switzerland	CHE	1.077221	8.372098e+06	39516.0
2	Germany	DEU	1.193867	8.266768e+07	348900.0
4	Spain	ESP	-0.008048	4.644396e+07	500210.0
5	France	FRA	0.407491	6.689611e+07	547557.0
6	Japan	JPN	-0.115284	1.269945e+08	364560.0
7	Greece	GRC	-0.687543	1.074674e+07	128900.0
8	Iran	IRN	1.148789	8.027743e+07	1628760.0
11	Nigeria	NGA	2.619034	1.859896e+08	910770.0
12	Qatar	QAT	3.495070	2.569804e+06	11610.0
14	India	IND	1.148215	1.324171e+09	2973190.0
15	World	WLD	1.181680	7.442136e+09	129733172.7

Filling NaN

The fillna function can “fill in” NA values with non-NA data in a couple of ways

In [14]:
```
country.fillna({'pop_growth':0, 'pop':100000000, 'Area':500000})
```

Out[14]:

	Name	Code	pop_growth	pop	Area
0	Brazil	BRA	0.817556	2.076529e+08	8358140.0
1	Switzerland	CHE	1.077221	8.372098e+06	39516.0
2	Germany	DEU	1.193867	8.266768e+07	348900.0
3	Denmark	DNK	0.834638	1.000000e+08	42262.0
4	Spain	ESP	-0.008048	4.644396e+07	500210.0
5	France	FRA	0.407491	6.689611e+07	547557.0
6	Japan	JPN	-0.115284	1.269945e+08	364560.0
7	Greece	GRC	-0.687543	1.074674e+07	128900.0
8	Iran	IRN	1.148789	8.027743e+07	1628760.0
9	Kuwait	KWT	2.924206	4.052584e+06	500000.0
10	Morocco	MAR	0.000000	3.527679e+07	446300.0
11	Nigeria	NGA	2.619034	1.859896e+08	910770.0
12	Qatar	QAT	3.495070	2.569804e+06	11610.0
13	Sweden	SWE	0.000000	9.903122e+06	407310.0
14	India	IND	1.148215	1.324171e+09	2973190.0
15	World	WLD	1.181680	7.442136e+09	129733172.7

In [15]:
```
country.fillna(method='ffill')
```

Out[15]:

	Name	Code	pop_growth	pop	Area
0	Brazil	BRA	0.817556	2.076529e+08	8358140.0
1	Switzerland	CHE	1.077221	8.372098e+06	39516.0
2	Germany	DEU	1.193867	8.266768e+07	348900.0
3	Denmark	DNK	0.834638	8.266768e+07	42262.0
4	Spain	ESP	-0.008048	4.644396e+07	500210.0
5	France	FRA	0.407491	6.689611e+07	547557.0
6	Japan	JPN	-0.115284	1.269945e+08	364560.0
7	Greece	GRC	-0.687543	1.074674e+07	128900.0
8	Iran	IRN	1.148789	8.027743e+07	1628760.0
9	Kuwait	KWT	2.924206	4.052584e+06	1628760.0
10	Morocco	MAR	2.924206	3.527679e+07	446300.0
11	Nigeria	NGA	2.619034	1.859896e+08	910770.0
12	Qatar	QAT	3.495070	2.569804e+06	11610.0
13	Sweden	SWE	3.495070	9.903122e+06	407310.0
14	India	IND	1.148215	1.324171e+09	2973190.0
15	World	WLD	1.181680	7.442136e+09	129733172.7

In [ ]:
```
from sklearn.preprocessing import Imputer

imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
imp.fit(country)
new_dataset = imp.transform(country)
new_dataset
#country.mean()
```

Dupplicates

In [17]:
```
my_data = pd.DataFrame({'Columns1':[1, 2, 2, 3, 4, 4] ,'Columns2':['a', 'a', 'a', 'b', 'b', 'b'], 'Columns3':['A', 'A', 'A', 'B', 'B', 'B']})
```

In [18]:
```
my_data.drop_duplicates(['Columns2'])
```

Out[18]:

	Columns1	Columns2	Columns3
0	1	a	A
3	3	b	B

Concatenating

In [19]:
```
my_source1 = pd.DataFrame([[0, 1, 13, 17, 17],
                           [0, 2, 14, 14, 15],
                           [0, 3, 17, 12, 20],
                           [0, 4, 12, 18, 19]])
my_source1[0] = ['babak', 'raha', 'sara', 'reza']
my_source1
```

Out[19]:

	0	1	2	3	4
0	babak	1	13	17	17
1	raha	2	14	14	15
2	sara	3	17	12	20
3	reza	4	12	18	19

In [20]:
```
my_source2 = pd.DataFrame([[0,1,13, 17],
                           [0,2,15, 20],
                           [0,3, 14, 19],
                           [0,4, 20, 19],
                           [0,5, 15, 18],
                           [0,6, 14, 12]])
my_source2[0]=['babak', 'baran', 'sara', 'arash', 'mahan', 'reza']
my_source2
```

Out[20]:

	0	1	2	3
0	babak	1	13	17
1	baran	2	15	20
2	sara	3	14	19
3	arash	4	20	19
4	mahan	5	15	18
5	reza	6	14	12

In [21]:
```
my_source1
```

Out[21]:

	0	1	2	3	4
0	babak	1	13	17	17
1	raha	2	14	14	15
2	sara	3	17	12	20
3	reza	4	12	18	19

In [22]:
```
my_source2
```

Out[22]:

	0	1	2	3
0	babak	1	13	17
1	baran	2	15	20
2	sara	3	14	19
3	arash	4	20	19
4	mahan	5	15	18
5	reza	6	14	12

In [23]:
```
my_concat = pd.concat([my_source1, my_source2], axis=0, ignore_index=True)
```

In [24]:
```
my_concat.drop([4], axis=1, inplace=True)
```

New Dataset

In [25]:
```
my_concat.drop_duplicates(inplace=True)
```

In [26]:
```
my_concat.reset_index(drop=True, inplace=True)
my_concat
```

Out[26]:

	0	1	2	3
0	babak	1	13	17
1	raha	2	14	14
2	sara	3	17	12
3	reza	4	12	18
4	baran	2	15	20
5	sara	3	14	19
6	arash	4	20	19
7	mahan	5	15	18
8	reza	6	14	12

Frequency counts for categorical data

In [27]:
```
smartphones = pd.read_csv('c://smartphones.csv')
#smartphones.describe()
smartphones.Capacity.value_counts()
```

Out[27]:
```
16     3
32     2
128    2
64     2
Name: Capacity, dtype: int64
```

Group and aggregate

In [28]:
```
cat_os = smartphones.groupby(smartphones['Company'])
cat_os.mean()
```

Out[28]:

	Capacity	Ram	Weight	inch
Company				
Apple	80.0	1.5	125.0	4.35
Google	128.0	4.0	143.0	5.00
HTC	64.0	4.0	170.0	5.70
Microsoft	32.0	3.0	150.0	5.20
Motorola	16.0	3.0	144.5	5.00
Samsung	40.0	3.0	147.0	5.45
Sony	16.0	2.0	180.0	5.50

Crosstab

> Crosstab or Cross Tabulation is used to aggregate and jointly display the distribution of two or more variables by tabulating their results one against the other in 2-dimensional grids.

In [29]:
```
pd.crosstab(smartphones.OS, smartphones.Capacity)
```

Out[29]:

Capacity	16	32	64	128
OS				
Android	3	0	2	1
ios	0	1	0	1
windows	0	1	0	0

Piovt Table

In [30]:
```
smartphones
```

Out[30]:

	Name	OS	Capacity	Ram	Weight	Company	inch
0	Galaxy S8	Android	64	4	149.0	Samsung	5.8
1	Lumia 950	windows	32	3	150.0	Microsoft	5.2
2	Xpreia L1	Android	16	2	180.0	Sony	5.5
3	iphone 7	ios	128	2	138.0	Apple	4.7
4	U Ultra	Android	64	4	170.0	HTC	5.7
5	Galaxy S5	Android	16	2	145.0	Samsung	5.1
6	iphone 5s	ios	32	1	112.0	Apple	4.0
7	Moto G5	Android	16	3	144.5	Motorola	5.0
8	Pixel	Android	128	4	143.0	Google	5.0

In [ ]:
```
pd.pivot_table(smartphones, index="Name", columns='Company', values='Ram')
```

Out[ ]:

Company	Apple	Google	HTC	Microsoft	Motorola	Samsung	Sony
Name							
Galaxy S5	NaN	NaN	NaN	NaN	NaN	2.0	NaN
Galaxy S8	NaN	NaN	NaN	NaN	NaN	4.0	NaN
Lumia 950	NaN	NaN	NaN	3.0	NaN	NaN	NaN
Moto G5	NaN	NaN	NaN	NaN	3.0	NaN	NaN
Pixel	NaN	4.0	NaN	NaN	NaN	NaN	NaN
U Ultra	NaN	NaN	4.0	NaN	NaN	NaN	NaN
Xpreia L1	NaN	NaN	NaN	NaN	NaN	NaN	2.0
iphone 5s	1.0	NaN	NaN	NaN	NaN	NaN	NaN
iphone 7	2.0	NaN	NaN	NaN	NaN	NaN	NaN

dumy variables

> scikit learn will not accept categorical features by default and need to encode categorical features numerically
> **pd.get_dummies()**: Convert categorical variable into dummy/indicator variables

In [ ]:
```
smartphones.rename(index=smartphones.Name, inplace=True)
smartphones.drop(['Name', 'Company'], axis=1, inplace=True)
smartphones
```

In [ ]:
```
smartphones_data = pd.get_dummies(smartphones)
smartphones_data.drop('OS_windows', axis=1, inplace=True)
smartphones_data
```

Normalize Data

In [ ]:
```
from sklearn.preprocessing import scale, normalize, minmax_scale
```

In [43]:
```
scale_data = minmax_scale(smartphones_data, feature_range=(0, 100))
```

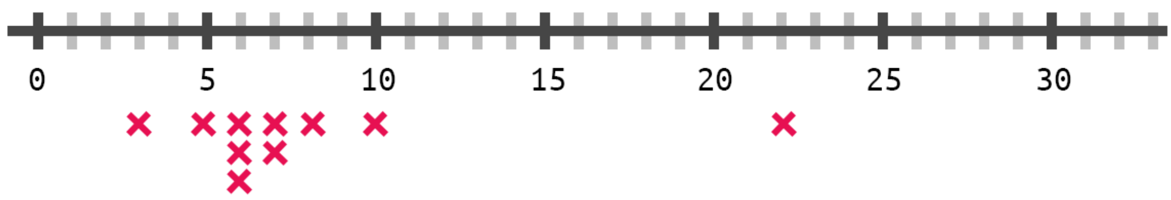
In [44]:
```
df_data = pd.DataFrame(scale_data,
                       index=smartphones_data.index,
                       columns=smartphones_data.columns)
df_data
```

Out[44]:

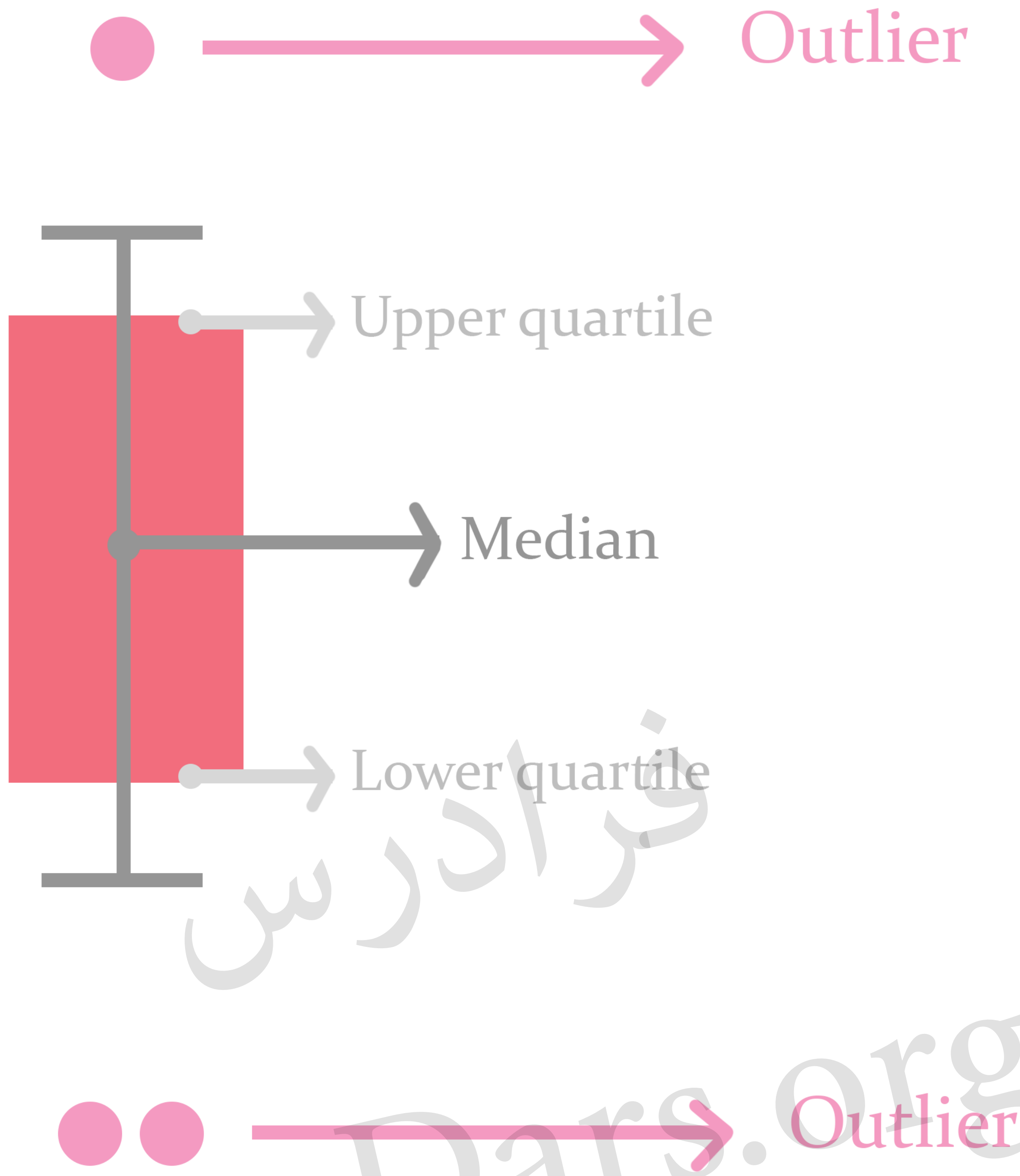
	Capacity	Ram	Weight	inch	OS_Android	OS_ios
Galaxy S8	42.857143	100.000000	54.411765	100.000000	100.0	0.0
Lumia 950	14.285714	66.666667	55.882353	66.666667	0.0	0.0
Xpreia L1	0.000000	33.333333	100.000000	83.333333	100.0	0.0
iphone 7	100.000000	33.333333	38.235294	38.888889	0.0	100.0
U Ultra	42.857143	100.000000	85.294118	94.444444	100.0	0.0
Galaxy S5	0.000000	33.333333	48.529412	61.111111	100.0	0.0
iphone 5s	14.285714	0.000000	0.000000	0.000000	0.0	100.0
Moto G5	0.000000	66.666667	47.794118	55.555556	100.0	0.0
Pixel	100.000000	100.000000	45.588235	55.555556	100.0	0.0

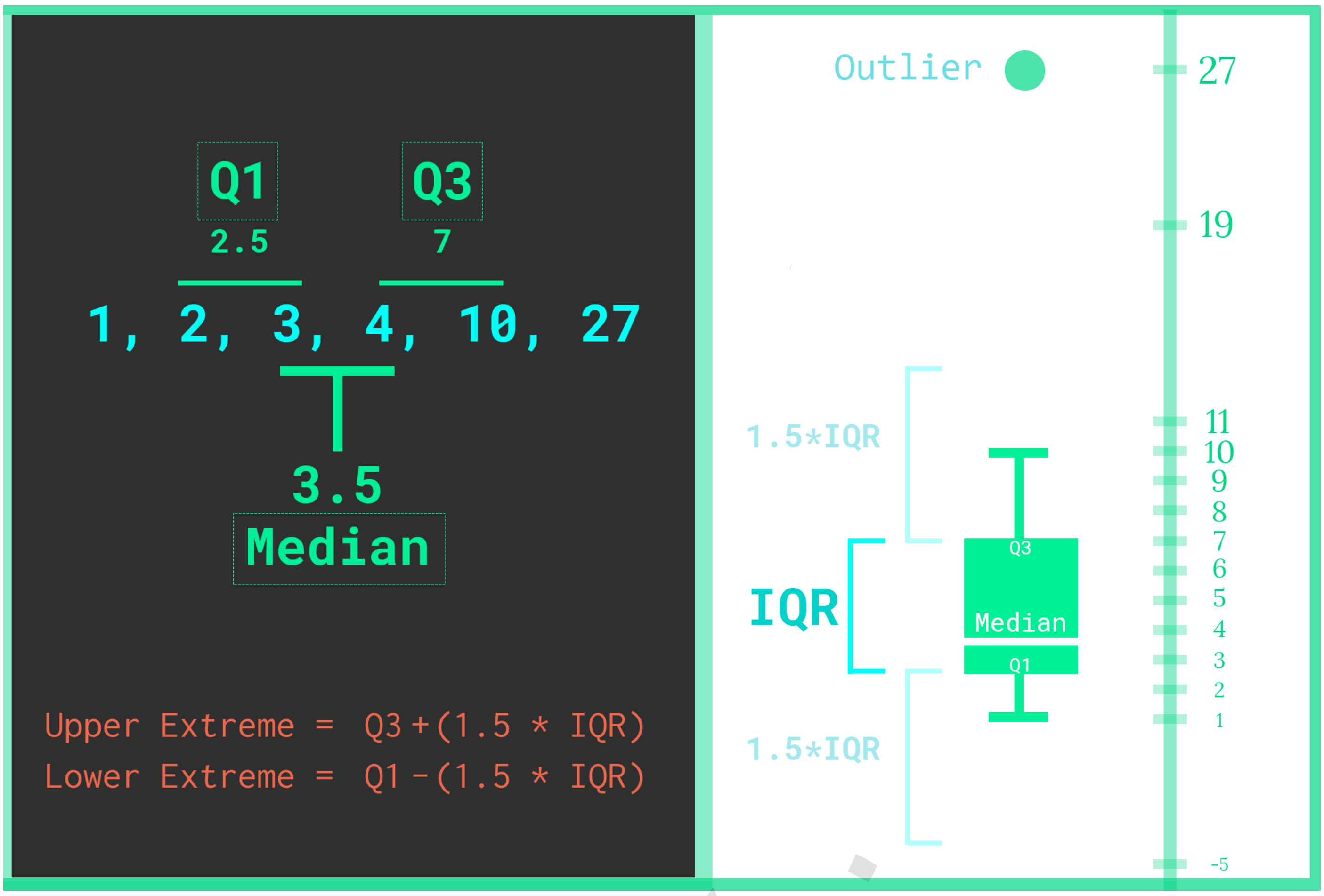
Outliers

Outlier



1	2	3	4	5	6	7	8	9	10
6	6	7	5	7	3	7	10	22	8

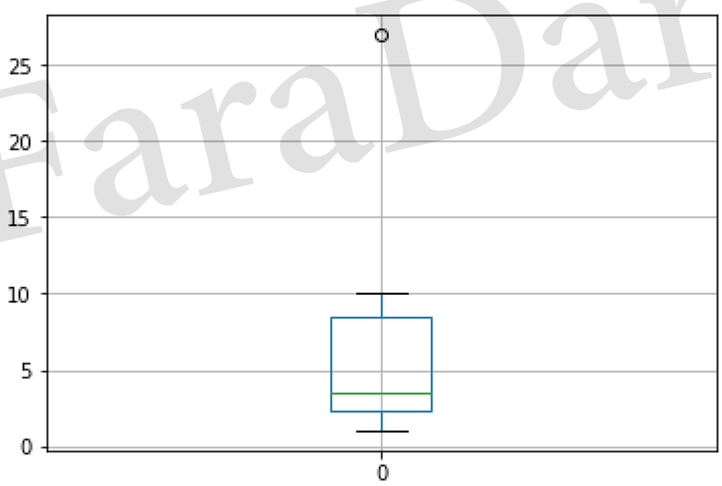

Outlier
Upper quartile
Median
Lower quartile
Outlier




```
In [45]: df = pd.DataFrame(np.array([1, 2, 3, 4, 10, 27]))
```

```
In [48]: df.quantile(0.75)
Out[48]: 0    8.5
         Name: 0.75, dtype: float64
```

```
In [49]: import matplotlib.pyplot as plt
         df.boxplot()
         plt.show()
```



q1 = 2.5 q3 = 7

IQR = 7 - 2.5 = 4.5

Upper extreme = 7 + (1.5 *4.5) = 13.75 Lower extreme = 2.5 - (1.5* 4.5) = -4.25