In [ ]:
```
import numpy as np
import pandas as pd
```

In [ ]:
```
df = pd.DataFrame({
    'Sex': ['M', 'F', 'F', 'D', '?'],
    'Age': [29, 30, 24, 290, 25],
})
df
```

Out[ ]:

	Sex	Age
0	M	29
1	F	30
2	F	24
3	D	290
4	?	25

The previous DataFrame doesn't have any "missing value", but clearly has invalid data. 290 doesn't seem like a valid age, and D and ? don't correspond with any known sex category. How can you clean these not-missing, but clearly invalid values then?

FINDING UNIQUE VALUES:

> The first step to clean invalid values is to notice them, then identify them and finally handle them appropriately (remove them, replace them, etc). Usually, for a "categorical" type of field (like Sex, which only takes values of a discrete set ('M', 'F')), we start by analyzing the variety of values present. For that, we use the unique() method:

In [ ]:
```
df['Sex'].unique()
```

Out[ ]:
```
array(['M', 'F', 'D', '?'], dtype=object)
```

In [ ]:
```
df['Sex'].value_counts()
```

Out[ ]:
```
F    2
D    1
?    1
M    1
Name: Sex, dtype: int64
```

Clearly if you see values like 'D' or '?', it'll immediately raise your attention. Now, what to do with them? Let's say you picked up the phone, called the survey company and they told you that 'D' was a typo and it should actually be F. You can use the replace function to replace these values:

In [ ]:
```
df['Sex'].replace('D', 'F')
```

Out[ ]:
```
0    M
1    F
2    F
3    F
4    ?
Name: Sex, dtype: object
```

It can accept a dictionary of values to replace. For example, they also told you that there might be a few 'N's, that should actually be 'M's:

In [ ]:
```
df['Sex'].replace({'D': 'F', 'N': 'M'})
```

Out[ ]:
```
0    M
1    F
2    F
3    F
4    ?
Name: Sex, dtype: object
```

We can also do something like this:

In [ ]:
```
df.replace({
    'Sex': {
        'D': 'F',
        'N': 'M'
    },
    'Age': {
        290: 29
    }
})
```

Out[ ]:

	Sex	Age
0	M	29
1	F	30
2	F	24
3	F	29
4	?	25

DUPLICATES:

Checking duplicate values is extremely simple. It'll behave differently between Series and DataFrames. Let's start with Series. As an example, let's say we're throwing a fancy party and we're inviting Ambassadors from Europe. But can only invite one ambassador per country. This is our original list, and as you can see, both the UK and Germany have duplicated ambassadors:

In [ ]:
```
ambassadors = pd.Series([
    'France',
    'United Kingdom',
    'United Kingdom',
    'Italy',
    'Germany',
    'Germany',
    'Germany',
], index=[
```

```
    'Gérard Araud',
    'Kim Darroch',
    'Peter Westmacott',
    'Armando Varricchio',
    'Peter Wittig',
    'Peter Ammon',
    'Klaus Scharioth '
])
ambassadors
```

Out[ ]:
```
Gérard Araud                 France
Kim Darroch           United Kingdom
Peter Westmacott      United Kingdom
Armando Varricchio             Italy
Peter Wittig                 Germany
Peter Ammon                  Germany
Klaus Scharioth              Germany
dtype: object
```

duplicated() tells us which values are duplicated. Here the *values* are the *Countries*

In [ ]:
```
ambassadors.duplicated()  #top down checking
```

Out[ ]:
```
Gérard Araud          False
Kim Darroch           False
Peter Westmacott       True
Armando Varricchio    False
Peter Wittig          False
Peter Ammon            True
Klaus Scharioth        True
dtype: bool
```

In [ ]:
```
ambassadors.duplicated(keep='last')  #bottom up checking
```

Out[ ]:
```
Gérard Araud          False
Kim Darroch            True
Peter Westmacott      False
Armando Varricchio    False
Peter Wittig           True
Peter Ammon            True
Klaus Scharioth       False
dtype: bool
```

In [ ]:
```
ambassadors.duplicated(keep=False)  #all repeated values are considered duplicates
```

Out[ ]:
```
Gérard Araud          False
Kim Darroch            True
Peter Westmacott       True
Armando Varricchio    False
Peter Wittig           True
Peter Ammon            True
Klaus Scharioth        True
dtype: bool
```

.drop_duplicates() does the same thing as before. Here, we can use stuff like **keep**

In [ ]:
```
ambassadors.drop_duplicates()
```

Out[ ]:
```
Gérard Araud                 France
Kim Darroch           United Kingdom
Armando Varricchio             Italy
Peter Wittig                 Germany
dtype: object
```

DUPLICATES IN DATAFRAMES:

In [ ]:
```
players = pd.DataFrame({
    'Name': [
        'Kobe Bryant',
        'LeBron James',
        'Kobe Bryant',
        'Carmelo Anthony',
        'Kobe Bryant',
    ],
    'Pos': [
        'SG',
        'SF',
        'SG',
        'SF',
        'SF'
    ]
})
players
```

Out[ ]:

	Name	Pos
0	Kobe Bryant	SG
1	LeBron James	SF
2	Kobe Bryant	SG
3	Carmelo Anthony	SF
4	Kobe Bryant	SF

In [ ]:
```
players.duplicated()
```

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

In [ ]:
```
players.duplicated(subset=['Name']) #instead of keep here
```

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

In [ ]:
```
players.drop_duplicates(subset=['Name'])
```

Out[ ]:

	Name	Pos
0	Kobe Bryant	SG
1	LeBron James	SF
3	Carmelo Anthony	SF

SPLITTING COLUMNS:

In [ ]:
```
df = pd.DataFrame({
    'Data': [
        '1987_M_US _1',
        '1990?_M_UK_1',
        '1992_F_US_2',
        '1970?_M_   IT_1',
        '1985_F_I  T_2'
]})
df
```

Out[ ]:

	Data
0	1987_M_US _1
1	1990?_M_UK_1
2	1992_F_US_2
3	1970?_M_ IT_1
4	1985_F_I T_2

splitting with "_" through "data"

In [ ]:
```
df['Data'].str.split('_')
```

Out[ ]:
```
0       [1987, M, US , 1]
1       [1990?, M, UK, 1]
2        [1992, F, US, 2]
3    [1970?, M,    IT, 1]
4      [1985, F, I  T, 2]
Name: Data, dtype: object
```

In [ ]:
```
df['Data'].str.split('_', expand=True) #CREATES A DATAFRAME OUT OF THE NEW THING WI

df = df['Data'].str.split('_', expand=True)
```

In [ ]:
```
df
```

Out[ ]:

	0	1	2	3
0	1987	M	US	1
1	1990?	M	UK	1
2	1992	F	US	2
3	1970?	M	IT	1
4	1985	F	I T	2

In [ ]:
```
df.columns = ['Year', 'Sex', 'Country', 'No Children']
df
```

Out[ ]:

	Year	Sex	Country	No Children
0	1987	M	US	1
1	1990?	M	UK	1
2	1992	F	US	2
3	1970?	M	IT	1
4	1985	F	I T	2

contains takes a regex/pattern as first value, so we need to escape the ? symbol as it has a special meaning for these patterns. Regular letters don't need escaping:

In [ ]:
```
df['Year'].str.contains('\?')
```

Out[ ]:
```
0    False
1     True
2    False
3     True
4    False
Name: Year, dtype: bool
```

In [ ]:
```
df['Country'].str.contains('U')
```

Out[ ]:
```
0     True
1     True
2     True
3    False
4    False
Name: Country, dtype: bool
```

In [ ]:
```
df['Country'].str.replace(' ', '') # to remove the I T to make it IT
```

Out[ ]:
```
0    US
1    UK
2    US
3    IT
4    IT
Name: Country, dtype: object
```