Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

```
Empty cells
Data in wrong format
Wrong data
Duplicates
```

In [ ]:
```
import pandas as pd
df = pd.read_csv('D:\Subject Materials\Python\Pandas\data1.csv')
#print(df.to_string())
print(df)
```

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

Note: By default, the dropna() method returns a new DataFrame, and will not change the original.

In [ ]:
```
df = pd.read_csv('D:\Subject Materials\Python\Pandas\data1.csv')
print(df.info())
```

In [ ]:
```
df = pd.read_csv('D:\Subject Materials\Python\Pandas\data1.csv')
new_df = df.dropna()  # Returns the Modified Data Frame, but no change in Origina
print(new_df)
```

In [ ]:
```
df = pd.read_csv('D:\Subject Materials\Python\Pandas\data1.csv')
df.dropna(inplace = True)
# Don't return any Data Frame, but modify's the Original Data Frame
# i.e  it will remove all rows containing NULL values from the original DataFrame
print(df.to_string())
```

Replace Empty Values

```
Another way of dealing with empty cells is to insert a new value instea
d.
This way you do not have to delete entire rows just because of some empt
y cells.
The fillna() method allows us to replace empty cells with a value:
For Example : Replace NULL values with the number 0:
```

In [ ]:
```
df = pd.read_csv('D:\Subject Materials\Python\Pandas\data1.csv')
df.fillna(0, inplace = True)
print(df)
print(df.info())
```

Replace Only For Specified Columns

```
The example above replaces all empty cells in the whole Data Frame.
To only replace empty values for one column, specify the column name for
the DataFrame:
```

In [ ]:
```
df["Calories"].fillna(130, inplace = True)
print(df)
```

Replace Using Mean, Median, or Mode

```
Replace Using Mean, Median, or Mode
Pandas uses the mean() median() and mode() methods to calculate the resp
ective values for a specified column:
```

In [ ]:
```
df = pd.read_csv('D:\Subject Materials\Python\Pandas\data1.csv')
#x = df["Calories"].mean()        # mean
#x = df["Calories"].median()      # median
x = df["Calories"].mode()[0]      # mode  i.e Mode = the value that appears most
df["Calories"].fillna(x, inplace = True)

print(df)
```

Data of Wrong Format

Cells with data of wrong format can make it difficult, or even impossible, to analyze data.

To fix it, you have two options: remove the rows, or convert all cells in the columns into the same format.

In [ ]:
```
df = pd.read_csv('D:\Subject Materials\Python\Pandas\data2.csv')
print(df)
```

In [ ]:
```
import pandas as pd

df['Date'] = pd.to_datetime(df['Date'])
print(df)
```

NaT (Not a Time)

Removing Rows

```
The result from the converting in the example above gave us a NaT value,
which can be handled as a NULL value, and we can remove the row by using
the dropna() method.

Remove rows with a NULL value in the "Date" column:
```

In [ ]:
```
df.dropna(subset=['Date'], inplace = True)
df
```

Pandas - Fixing Wrong Data

Wrong Data

```
"Wrong data" does not have to be "empty cells" or "wrong format", it can
just be wrong, like if someone registered "199" instead of "1.99".

Sometimes you can spot wrong data by looking at the data set, because yo
u have an expectation of what it should be.

If you take a look at our data set, you can see that in row 7, the durat
ion is 450, but for all the other rows the duration is between 30 and 6
0.

It doesn't have to be wrong, but taking in consideration that this is th
e data set of someone's workout sessions, we conclude with the fact that
this person did not work out in 450 minutes.
```

In [ ]:
```
df.loc[3, 'Duration'] = 45
df
```

```
For small data sets you might be able to replace the wrong data one by o
```

ne, but not for big data sets.

To replace wrong data for larger data sets you can create some rules, e. g. set some boundaries for legal values, and replace any values that are outside of the boundaries.

Example

Loop through all values in the "Duration" column.

If the value is higher than 120, set it to 120:

In [ ]:
```
for x in df.index:
  if df.loc[x, "Duration"] > 45:
    df.loc[x, "Duration"] = 120

df
```

Removing Rows

Another way of handling wrong data is to remove the rows that contains wrong data.

This way you do not have to find out what to replace them with, and there is a good chance you do not need them to do your analyses.

In [ ]:
```
for x in df.index:
  if df.loc[x, "Duration"] > 45:
    df.drop(x, inplace = True)

df
```

Pandas - Removing Duplicates

Discovering Duplicates

Duplicate rows are rows that have been registered more than one time.
To discover duplicates, we can use the duplicated() method.
The duplicated() method returns a Boolean values for each row(True/False)

Removing Duplicates

To remove duplicates, use the drop_duplicates() method.

In [ ]:
```
print(df.duplicated())
```

In [ ]:
```
df.drop_duplicates(inplace = True)
print(df)
```