Data Cleansing & Pre-Processing

1. Libraries & Data Import

Only two packages are required for data cleansing at first. The next step is to import the file.

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

>>> df1 = pd.read_csv('file.csv')
>>> df2 = pd.read_excel('file.xlsx',
...                      sheet_name = 0,
...                      header = 1,
...                      na_values = ['#NV', 'Err'])
```

2. Inspect Data Set

Draw a random sample of rows in order to get an idea of the content of the data frame.

```
>>> a = pd.DataFrame({
...      'C0': ['1', '2', '3', '4'],
...      'C1': [.321, .654, .987, .741],
...      'C2': ['abc', 'def', 'ghi', 'jkl']},
...      index=['obj0', 'obj1', 'obj2', 'obj3'])
>>> a.sample(n=2)
        C0      C1      C2
obj3     4   0.741     jkl
obj1     2   0.654     def
>>> a. describe(include='all')
        C0      C1      C2
count    4   4.000       4
unique   4     NaN       4
top      3     NaN     jkl
freq     1     NaN       1
...    ...     ...     ...
```

3. Data Types

If columns are not correctly recognized as numeric data types, this must be corrected.

```
>>> a.dtypes # C0 should be int64 but is object
C0      object
C1     float64
C2      object
dtype: object
>>> a['C0'] = pd.to_numeric(a['C0'])
```

4. Missings

```
>>> b = pd.DataFrame(
...     {'Name': ['Anna', 'Bill', 'Charly', 'Diana'],
...      'Job': ['Data Scientist',
...              'DATA SCIENTIST',
...              'Data Engineer ',
...              ' Data Science'],
...      'Age': [26, np.nan, 30, 32],
...      'CompCar': [np.nan, np.nan, np.nan, 'BMW']},
...     index=['emp01', 'emp02', 'emp03', 'emp04'])
```

4.1 Missing Statistics

How much columns contain missing values (% of columns)?
How much missing data is in the data frame (% of cells)?

```
>>> len(b.columns[b.isna().any()])/len(b.columns)
0.5
>>> b.isnull().sum().sum()/np.product(b.shape)
0.25
```

4.2 Missing Types

It must be considered which origin missing values (probably) have. A distinction must be made between (a) non-existent and (b) non-recorded values. The processing is different here.

> *(a) Non-existent*
> *Example: Employee has no company car, so no manufacturer is recorded.*
> *Handling: Keep NaN; maybe replaced by specific value, e.g. (0, 'nothing')*
> *(b) Non-recorded*
> *Example: No information about age of employee*
> *Handling: Imputation*

4.3 Imputation

1) Use mean/median of column (numerical features only)

```
>>> b['Age'] = b['Age'].fillna(b['Age'].mean())
```

2) Use most frequent value in column (numerical and categorical features)

```
>>> b['Age'] = b['Age'].fillna(b['Age'].mode()[0])
```

3) Hot-deck imputation; use a random value of one of the other objects (numerical and categorical features)

```
>>> b.loc[b['Age'].isna(), ['Age']] = (
...     np.random.choice(np.unique(b['Age'].dropna()),
...                      b['Age'].isna().sum())
...    )
```

More sophisticated approaches are among others:

4) Regression and stochastic regression imputation
5) Imputation via k-NN
6) Deep Learning imputation
7) Multivariate Imputation by Chained Equations (MICE)

Some methods (e.g. stochastic regression, hot-deck, MICE) can be used with multiple imputation. Here, the uncertainty is incorporated by using not only a single but multiple imputation values.

5. Inconsistencies

The meaning of categorical data can be corrupted by incorrect spelling or useless characters. These irregularities must be removed.

5.1 Upper and lower cases

```
>>> b['Job'] = b['Job'].str.lower()
```

5.2 Leading and trailing white spaces

```
>>> b['Job'] = b['Job'].str.strip()
```

5.3 Notation similarities

The meaning of expressions can be the same despite different spellings. Here it is possible to make a quantitative comparison of strings and to visualize the permutations. Then it must be decided whether and if so how certain values are adapted/overwritten.

```
>>> from fuzzywuzzy import process
>>> import seaborn as sns; sns.set();
>>> def SpellCheck(df, col):
...     instances=np.unique(df[col])
...     dt=[(col, object),('score', int)]
...     result=np.zeros(shape=(len(instances),
...                            len(instances)))
...     for i, inst in enumerate(instances):
...         result[i]=np.sort(np.array(
...             process.extract(inst,instances),
...             dtype = dt),
...                           order = col)['score']
...     result=pd.DataFrame(result,
...                         index=instances,
...                         columns=instances)
...     return result
>>> # Adjust vmin and vmax to accepted similarities
>>> ax = sns.heatmap(mapping, vmin=60, vmax=100,
...                  annot=True, fmt=".0f",
...                  cmap="Reds")
```