Data Science 01: Příprava a čištění dat (Data Preparation / Cleaning)

Získávání dat (Data Acquisition)

Dataset TimeLaps, vzniklý na základě časosběrného monitorování robotických zdicích prací, uložen ve formátu CSV

In [1]:
```
# Instalace potřebných knihoven
#%pip install pandas
#%pip install numpy
```

In [2]:
```
# Import potřebných knihoven
import pandas as pd
import numpy as np
```

In [3]:
```
# Soubor je načten a přiřazen do proměnné ‚df‘
other_path = '../../data/00_Raw/timelaps.csv'
df = pd.read_csv(other_path, header=0)
```

In [4]:
```
# Zobrazení prvních 5 řádků datasetu
print('Prvních 5 řádků datového rámce')
df.head(5)
```

```
Prvních 5 řádků datového rámce
```

Out[4]:

	ID	TYPE	START	CAL	WALL	END
0	1	CORNER	13:52:18	13:52:24	13:52:41	13:52:59
1	2	HALF	13:52:59	13:53:02	13:53:19	13:53:35
2	3	BASIC	13:53:35	13:55:21	13:55:43	13:56:00
3	4	BASIC	13:56:00	13:56:06	13:56:20	13:56:37
4	5	BASIC	13:56:36	13:56:41	13:56:55	13:58:11

Přidání nebo změna názvů sloupců

In [5]:
```
# Tvorba názvů sloupců
headers = ['id', 'type_brick', 'time_start', 'time_verif', 'time_dest', 'time_end']
print('Sloupce\n', headers)
```

```
Sloupce
 ['id', 'type_brick', 'time_start', 'time_verif', 'time_dest', 'time_end']
```

In [6]:
```
# Nahrazení názvů sloupců a následná kontrola datového rámce
df.columns = headers
df.head()
```

Out[6]:

	id	type_brick	time_start	time_verif	time_dest	time_end
0	1	CORNER	13:52:18	13:52:24	13:52:41	13:52:59
1	2	HALF	13:52:59	13:53:02	13:53:19	13:53:35
2	3	BASIC	13:53:35	13:55:21	13:55:43	13:56:00
3	4	BASIC	13:56:00	13:56:06	13:56:20	13:56:37
4	5	BASIC	13:56:36	13:56:41	13:56:55	13:58:11

Analýza chybějících hodnot v datové sadě

In [7]:
```
# Nahrazení symbolu „?“ chybějící hodnotou (NaN)
df = df.replace('?', np.nan)
df.head()
```

Out[7]:

	id	type_brick	time_start	time_verif	time_dest	time_end
0	1	CORNER	13:52:18	13:52:24	13:52:41	13:52:59
1	2	HALF	13:52:59	13:53:02	13:53:19	13:53:35
2	3	BASIC	13:53:35	13:55:21	13:55:43	13:56:00
3	4	BASIC	13:56:00	13:56:06	13:56:20	13:56:37
4	5	BASIC	13:56:36	13:56:41	13:56:55	13:58:11

In [8]:
```
# Logická hodnota ‚True‘ indikuje přítomnost chybějící hodnoty, zatímco ‚False‘ označuje její
missing_data = df.isnull()
missing_data.head(5)
```

Out[8]:

	id	type_brick	time_start	time_verif	time_dest	time_end
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False

In [9]:
```
# Výpočet počtu chybějících hodnot v jednotlivých sloupcích datového rámce
for column in missing_data.columns.values.tolist():
    print(f'{missing_data[column].value_counts()}\n')
```

```
id
False    136
Name: count, dtype: int64

type_brick
False    136
Name: count, dtype: int64

time_start
False    136
Name: count, dtype: int64

time_verif
False    136
Name: count, dtype: int64

time_dest
False    136
Name: count, dtype: int64

time_end
False    136
Name: count, dtype: int64
```

Práce s chybějícími daty

Jak pracovat s chybějícími daty?

1. Odstranění dat:
   a. Odstranění celého řádku
   b. Odstranění celého sloupce
2. Nahrazení dat:
   a. Nahrazení průměrnou hodnotou
   b. Nahrazení nejčastější hodnotou (frekvencí)
   c. Nahrazení na základě jiných funkcí

Výpočty a následné úpravy dat

1. Change "type_brick" data to category values:
   a. BASIC -> 1
   b. CORNER -> 2
   c. HALF -> 3
   d. END -> 4
2. Trasfeer values:
   a. time_start (text: HH:MM:SS) -> time_start(int: SS)
   b. time_verif (text: HH:MM:SS) -> time_verif(int: SS)
   c. time_dest (text: HH:MM:SS) -> time_dest(int: SS)
   d. time_end (text: HH:MM:SS) -> time_end(int: SS)
3. Calculate time cyclus:
   a. time_verif = time_verif - time_start
   b. time_dest = time_dest - time_start
   c. time_end = time_end - time_start
   d. time_start = 0

In [10]:
```
# Konverze časové hodnoty
def time_convert(x):
    h,m,s = map(int,x.split(':'))
    return (h*60+m)*60+s
```

In [11]:
```
df['time_start_sec'] = df.time_start.apply(time_convert)
df['time_verif_sec'] = df.time_verif.apply(time_convert)
df['time_dest_sec'] = df.time_dest.apply(time_convert)
df['time_end_sec'] = df.time_end.apply(time_convert)
df.head()
```

Out[11]:

	id	type_brick	time_start	time_verif	time_dest	time_end	time_start_sec	time_verif_sec	time_dest
0	1	CORNER	13:52:18	13:52:24	13:52:41	13:52:59	49938	49944	49
1	2	HALF	13:52:59	13:53:02	13:53:19	13:53:35	49979	49982	49
2	3	BASIC	13:53:35	13:55:21	13:55:43	13:56:00	50015	50121	50
3	4	BASIC	13:56:00	13:56:06	13:56:20	13:56:37	50160	50166	50
4	5	BASIC	13:56:36	13:56:41	13:56:55	13:58:11	50196	50201	50

Kontrola a úprava formátu dat

In [12]:
```
# Kontrola datového typu
df.dtypes
```

Out[12]:
```
id                int64
type_brick       object
time_start       object
time_verif       object
time_dest        object
time_end         object
time_start_sec    int64
time_verif_sec    int64
time_dest_sec     int64
time_end_sec      int64
dtype: object
```

In [13]:
```
# úprava formátu dat: Categories -> Int
df['type'] = df['type_brick'].replace(['CORNER', 'HALF', 'BASIC', 'END'], [2, 3, 1, 4]).infer_
df.head()
```

```
C:\Users\Vjačeslav Usmanov\AppData\Local\Temp\ipykernel_23504\716486450.py:2: FutureWarning: D
owncasting behavior in `replace` is deprecated and will be removed in a future version. To ret
ain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the fut
ure behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df['type'] = df['type_brick'].replace(['CORNER', 'HALF', 'BASIC', 'END'], [2, 3, 1, 4]).infe
r_objects(copy=False)
```

Out[13]:

	id	type_brick	time_start	time_verif	time_dest	time_end	time_start_sec	time_verif_sec	time_dest
0	1	CORNER	13:52:18	13:52:24	13:52:41	13:52:59	49938	49944	49
1	2	HALF	13:52:59	13:53:02	13:53:19	13:53:35	49979	49982	49
2	3	BASIC	13:53:35	13:55:21	13:55:43	13:56:00	50015	50121	50
3	4	BASIC	13:56:00	13:56:06	13:56:20	13:56:37	50160	50166	50
4	5	BASIC	13:56:36	13:56:41	13:56:55	13:58:11	50196	50201	50

Výpočet doby pracovního cyklu

In [14]:
```
df['start_to_verif'] = df['time_verif_sec'] - df['time_start_sec']
df['verif_to_dest'] = df['time_dest_sec'] - df['time_verif_sec']
df['dest_to_end'] = df['time_end_sec'] - df['time_dest_sec']
df['total_time'] = df['time_end_sec'] - df['time_start_sec']
df.head()
```

Out[14]:

	id	type_brick	time_start	time_verif	time_dest	time_end	time_start_sec	time_verif_sec	time_dest
0	1	CORNER	13:52:18	13:52:24	13:52:41	13:52:59	49938	49944	49
1	2	HALF	13:52:59	13:53:02	13:53:19	13:53:35	49979	49982	49
2	3	BASIC	13:53:35	13:55:21	13:55:43	13:56:00	50015	50121	50
3	4	BASIC	13:56:00	13:56:06	13:56:20	13:56:37	50160	50166	50
4	5	BASIC	13:56:36	13:56:41	13:56:55	13:58:11	50196	50201	50

In [15]:
```
df.describe()
```

Out[15]:

	id	time_start_sec	time_verif_sec	time_dest_sec	time_end_sec	type	start_to_veri
count	136.000000	136.000000	136.000000	136.000000	136.000000	136.000000	136.00000
mean	68.500000	53193.125000	53202.507353	53217.352941	53240.102941	1.250000	9.38235
std	39.403892	2905.472855	2904.759721	2903.065064	2895.630315	0.737865	9.24807
min	1.000000	48678.000000	48684.000000	48704.000000	48724.000000	1.000000	2.00000
25%	34.750000	50543.250000	50549.750000	50567.000000	50586.750000	1.000000	6.00000
50%	68.500000	51772.000000	51781.000000	51793.000000	51805.000000	1.000000	8.00000
75%	102.250000	55851.750000	55859.250000	55877.250000	55903.750000	1.000000	10.00000
max	136.000000	57450.000000	57467.000000	57473.000000	57482.000000	4.000000	106.00000

Identifikace a odstranění odlehlých hodnot

In [16]:
```
# Nastavení horní a dolní meze pro odlehlé hodnoty
low_limit = 0.10
hi_limit =  0.90

q_low = df['total_time'].quantile(low_limit)
q_hi  = df['total_time'].quantile(hi_limit)

df = df[(df['total_time'] < q_hi) & (df['total_time'] > q_low)]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 106 entries, 0 to 135
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              106 non-null    int64
 1   type_brick      106 non-null    object
 2   time_start      106 non-null    object
 3   time_verif      106 non-null    object
 4   time_dest       106 non-null    object
 5   time_end        106 non-null    object
 6   time_start_sec  106 non-null    int64
 7   time_verif_sec  106 non-null    int64
 8   time_dest_sec   106 non-null    int64
 9   time_end_sec    106 non-null    int64
 10  type            106 non-null    int64
 11  start_to_verif  106 non-null    int64
 12  verif_to_dest   106 non-null    int64
 13  dest_to_end     106 non-null    int64
 14  total_time      106 non-null    int64
dtypes: int64(10), object(5)
memory usage: 13.2+ KB
```

In [17]:
```
df.describe()
```

Out[17]:

	id	time_start_sec	time_verif_sec	time_dest_sec	time_end_sec	type	start_to_veri
count	106.000000	106.000000	106.000000	106.00000	106.000000	106.000000	106.00000
mean	69.622642	53179.698113	53188.084906	53203.00000	53220.660377	1.292453	8.38679
std	37.319586	2881.268663	2881.518674	2880.72757	2879.756921	0.780317	3.43253
min	1.000000	48678.000000	48684.000000	48704.00000	48724.000000	1.000000	2.00000
25%	39.250000	50539.750000	50547.250000	50565.00000	50580.250000	1.000000	6.00000
50%	67.500000	51736.500000	51745.500000	51757.00000	51772.500000	1.000000	8.00000
75%	102.750000	55879.250000	55885.750000	55905.75000	55921.250000	1.000000	10.00000
max	136.000000	57450.000000	57467.000000	57473.00000	57482.000000	4.000000	21.00000

Výpočet nejistoty

In [18]:
```
def measurement_uncertainty(df, column, delta_t=1, k=2):
    """
    Výpočet nejistoty měření z časosběrných dat.

    Parametry:
    df       : pandas DataFrame s daty
    column   : název sloupce s měřením (např. čas v s)
    delta_t  : časové rozlišení přístroje (s)
    k        : koeficient rozšíření (default k=2)

    Návratová hodnota:
    dict s výsledky
    """

    data = df[column].dropna()

    n = len(data)
    mean = data.mean()
```

```
    s = data.std(ddof=1)

    # Typ A
    u_A = s / np.sqrt(n)

    # Typ B (kvantizace)
    u_B_single = (delta_t / 2) / np.sqrt(3)
    u_B_mean = u_B_single / np.sqrt(n)

    # Kombinovaná
    u_c = np.sqrt(u_A**2 + u_B_mean**2)

    # Rozšířená
    U = k * u_c

    return {
        "serie": column,
        "n": n,
        "mean": mean,
        "std_dev": s,
        "u_A": u_A,
        "u_B_single": u_B_single,
        "u_B_mean": u_B_mean,
        "u_c": u_c,
        "U": U
    }

def report(result):
    print(f"{result['serie']}:")
    print(f"Kombinovaná nejistota: ({result['mean']:.2f} ± {result['u_c']:.2f})")
    print(f"Rozšířená nejistota: ({result['mean']:.2f} ± {result['U']:.2f}) s (k=2, 95%)")
    print(f"Nejistota Typ A (stochastická): {result['u_A']:.4f}")
    print(f"Nejistota Typ B (přístrojová / 1 měření): {result['u_B_single']:.4f}")
    print(f"Nejistota Typ B (přístrojová / průměr): {result['u_B_mean']:.4f}")
    print()
```

In [19]:

```
result = measurement_uncertainty(df, 'total_time', delta_t=1)
report(result)

result = measurement_uncertainty(df, 'start_to_verif', delta_t=1)
report(result)

result = measurement_uncertainty(df, 'verif_to_dest', delta_t=1)
report(result)

result = measurement_uncertainty(df, 'dest_to_end', delta_t=1)
report(result)
```

```
total_time:
Kombinovaná nejistota: (40.96 ± 0.62)
Rozšířená nejistota: (40.96 ± 1.24) s (k=2, 95%)
Nejistota Typ A (stochastická): 0.6176
Nejistota Typ B (přístrojová / 1 měření): 0.2887
Nejistota Typ B (přístrojová / průměr): 0.0280

start_to_verif:
Kombinovaná nejistota: (8.39 ± 0.33)
Rozšířená nejistota: (8.39 ± 0.67) s (k=2, 95%)
Nejistota Typ A (stochastická): 0.3334
Nejistota Typ B (přístrojová / 1 měření): 0.2887
Nejistota Typ B (přístrojová / průměr): 0.0280

verif_to_dest:
Kombinovaná nejistota: (14.92 ± 0.34)
Rozšířená nejistota: (14.92 ± 0.67) s (k=2, 95%)
Nejistota Typ A (stochastická): 0.3349
Nejistota Typ B (přístrojová / 1 měření): 0.2887
Nejistota Typ B (přístrojová / průměr): 0.0280

dest_to_end:
Kombinovaná nejistota: (17.66 ± 0.60)
Rozšířená nejistota: (17.66 ± 1.20) s (k=2, 95%)
Nejistota Typ A (stochastická): 0.5993
Nejistota Typ B (přístrojová / 1 měření): 0.2887
Nejistota Typ B (přístrojová / průměr): 0.0280
```

Export datové sady do formátu CSV

In [20]:
```
df.to_csv("../../data/01_DataScience/clean_timelaps.csv", index=False)
```

Read/Save Other Data Formats

Data Formate	Read	Save
csv	pd.read_csv()	df.to_csv()
json	pd.read_json()	df.to_json()
excel	pd.read_excel()	df.to_excel()
hdf	pd.read_hdf()	df.to_hdf()
sql	pd.read_sql()	df.to_sql()

Autor / Organizace / Datum

Vjačeslav Usmanov, ČVUT v Praze, Fakulta stavební

Přehled změn

Datum (YYYY-MM-DD)	Verze	Autor změny	Popis změny
2026-01-18	1.1	Vjačeslav Usmanov	added DS_01_Data_Cleaning.ipnyb
2026-02-10	1.2	Vjačeslav Usmanov	changed DS_01_Data_Cleaning.ipnyb