Workshop 1: Håndtering af Manglende Data

Fag: AI & Data, 2. Semester

Gruppemedlemmer: Peter & Jonas

Dato: 14-04-2025

Trin 0: Import af Nødvendige Biblioteker

Først importerer vi alle de Python-biblioteker, vi forventer at få brug for til dataindlæsning, manipulation, visualisering, imputering, modellering og databaseinteraktion. Dette inkluderer `pandas` til datahåndtering, `numpy` til numeriske operationer, `matplotlib` og `seaborn` til visualisering, `sklearn` til imputering og maskinlæring, samt `sqlite3` til databasehåndtering.

In [27]:
```
# Datahåndtering
import pandas as pd
import numpy as np

# Visualisering
import matplotlib.pyplot as plt
import seaborn as sns

# Imputering
from sklearn.impute import SimpleImputer, KNNImputer

# Preprocessing og Model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Database
import sqlite3

plt.style.use('seaborn-v0_8-whitegrid')
```

Trin 1: Indlæsning af Datasæt

Vi starter med at indlæse det valgte datasæt, `horse.csv`. Vi bruger `pandas`' `read_csv` funktion.

In [28]:
```
# Definer filsti
file_path = 'horse.csv'

df_original = pd.read_csv(file_path)

print(f"Datasættet har {df_original.shape[0]} rækker og {df_original.shape[1]} k

# Vis de første par rækker for at få et indtryk af data
print(df_original.head())
```

```
Datasættet har 299 rækker og 28 kolonner.
  surgery    age  hospital_number  rectal_temp  pulse  respiratory_rate  \
0      no  adult           530101         38.5   66.0              28.0
1     yes  adult           534817         39.2   88.0              20.0
2      no  adult           530334         38.3   40.0              24.0
3     yes  young          5290409         39.1  164.0              84.0
4      no  adult           530255         37.3  104.0              35.0

  temp_of_extremities peripheral_pulse mucous_membrane capillary_refill_time  \
0                cool          reduced             NaN            more_3_sec
1                 NaN              NaN   pale_cyanotic            less_3_sec
2              normal           normal       pale_pink            less_3_sec
3                cold           normal   dark_cyanotic            more_3_sec
4                 NaN              NaN   dark_cyanotic            more_3_sec

   ... packed_cell_volume total_protein abdomo_appearance abdomo_protein  \
0  ...               45.0           8.4               NaN            NaN
1  ...               50.0          85.0            cloudy            2.0
2  ...               33.0           6.7               NaN            NaN
3  ...               48.0           7.2     serosanguious            5.3
4  ...               74.0           7.4               NaN            NaN

      outcome  surgical_lesion lesion_1 lesion_2  lesion_3  cp_data
0        died               no    11300        0         0       no
1  euthanized               no     2208        0         0       no
2       lived               no        0        0         0      yes
3        died              yes     2208        0         0      yes
4        died               no     4300        0         0       no

[5 rows x 28 columns]
```

Trin 2: Indledende Dataudforskning og Identifikation af Manglende Data

Nu udforsker vi datasættet for at forstå dets struktur, datatyper og især omfanget og placeringen af manglende data.

Vi vil bruge metoder som `info()`, `describe()`, `isnull().sum()` og visualiseringer.

In [29]:
```python
# Tjekker om DataFrame blev indlæst korrekt
print("Grundlæggende Information")
df_original.info()

print("\n Deskriptiv Statistik (Numeriske Kolonner)")
print(df_original.describe())

print("\n Deskriptiv Statistik (Kategoriske Kolonner)")
print(df_original.describe(include='object'))

print("\n Antal Manglende Værdier pr. Kolonne")
missing_counts = df_original.isnull().sum()
print(missing_counts[missing_counts > 0].sort_values(ascending=False))

print("\n Procentdel Manglende Værdier pr. Kolonne")
missing_percentage = (df_original.isnull().sum() / len(df_original)) * 100
print(missing_percentage[missing_percentage > 0].sort_values(ascending=False))
```

Grundlæggende Information

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 28 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   surgery                299 non-null    object 
 1   age                    299 non-null    object 
 2   hospital_number        299 non-null    int64  
 3   rectal_temp            239 non-null    float64
 4   pulse                  275 non-null    float64
 5   respiratory_rate       241 non-null    float64
 6   temp_of_extremities    243 non-null    object 
 7   peripheral_pulse       230 non-null    object 
 8   mucous_membrane        252 non-null    object 
 9   capillary_refill_time  267 non-null    object 
 10  pain                   244 non-null    object 
 11  peristalsis            255 non-null    object 
 12  abdominal_distention   243 non-null    object 
 13  nasogastric_tube       195 non-null    object 
 14  nasogastric_reflux     193 non-null    object 
 15  nasogastric_reflux_ph  53 non-null     float64
 16  rectal_exam_feces      197 non-null    object 
 17  abdomen                181 non-null    object 
 18  packed_cell_volume     270 non-null    float64
 19  total_protein          266 non-null    float64
 20  abdomo_appearance      134 non-null    object 
 21  abdomo_protein         101 non-null    float64
 22  outcome                299 non-null    object 
 23  surgical_lesion        299 non-null    object 
 24  lesion_1               299 non-null    int64  
 25  lesion_2               299 non-null    int64  
 26  lesion_3               299 non-null    int64  
 27  cp_data                299 non-null    object 
dtypes: float64(7), int64(4), object(17)
memory usage: 65.5+ KB
```

Deskriptiv Statistik (Numeriske Kolonner)

	hospital_number	rectal_temp	pulse	respiratory_rate
count	2.990000e+02	239.000000	275.000000	241.000000
mean	1.087733e+06	38.168619	72.000000	30.460581
std	1.532032e+06	0.733744	28.646219	17.666102
min	5.184760e+05	35.400000	30.000000	8.000000
25%	5.289040e+05	37.800000	48.000000	18.000000
50%	5.303010e+05	38.200000	64.000000	25.000000
75%	5.347360e+05	38.500000	88.000000	36.000000
max	5.305629e+06	40.800000	184.000000	96.000000

	nasogastric_reflux_ph	packed_cell_volume	total_protein
count	53.000000	270.000000	266.000000
mean	4.707547	46.307407	24.274436
std	1.982311	10.436743	27.364194
min	1.000000	23.000000	3.300000
25%	3.000000	38.000000	6.500000
50%	5.000000	45.000000	7.500000
75%	6.500000	52.000000	56.750000
max	7.500000	75.000000	89.000000

	abdomo_protein	lesion_1	lesion_2	lesion_3
count	101.000000	299.000000	299.000000	299.000000

```
mean         3.039604   3659.709030    90.528428     7.387960
std          1.967947   5408.472421   650.637139   127.749768
min          0.100000      0.000000     0.000000     0.000000
25%          2.000000   2111.500000     0.000000     0.000000
50%          2.300000   2322.000000     0.000000     0.000000
75%          3.900000   3209.000000     0.000000     0.000000
max         10.100000  41110.000000  7111.000000  2209.000000

 Deskriptiv Statistik (Kategoriske Kolonner)
       surgery    age temp_of_extremities peripheral_pulse mucous_membrane  \
count      299    299                 243              230             252
unique       2      2                   4                4               6
top        yes  adult                cool           normal     normal_pink
freq       180    275                 108              114              79

       capillary_refill_time       pain peristalsis abdominal_distention  \
count                    267        244         255                   243
unique                     3          5           4                     4
top               less_3_sec  mild_pain  hypomotile                  none
freq                     187         67         127                    75

       nasogastric_tube nasogastric_reflux rectal_exam_feces        abdomen  \
count               195                193               197            181
unique                3                  3                 4              5
top              slight               none            absent  distend_large
freq                101                119                79             78

       abdomo_appearance outcome surgical_lesion cp_data
count                134     299             299     299
unique                 3       3               2       2
top                cloudy   lived             yes      no
freq                  47     178             190     200

 Antal Manglende Værdier pr. Kolonne
nasogastric_reflux_ph    246
abdomo_protein           198
abdomo_appearance        165
abdomen                  118
nasogastric_reflux       106
nasogastric_tube         104
rectal_exam_feces        102
peripheral_pulse          69
rectal_temp               60
respiratory_rate          58
temp_of_extremities       56
abdominal_distention      56
pain                      55
mucous_membrane           47
peristalsis               44
total_protein             33
capillary_refill_time     32
packed_cell_volume        29
pulse                     24
dtype: int64

 Procentdel Manglende Værdier pr. Kolonne
nasogastric_reflux_ph    82.274247
abdomo_protein           66.220736
abdomo_appearance        55.183946
abdomen                  39.464883
```

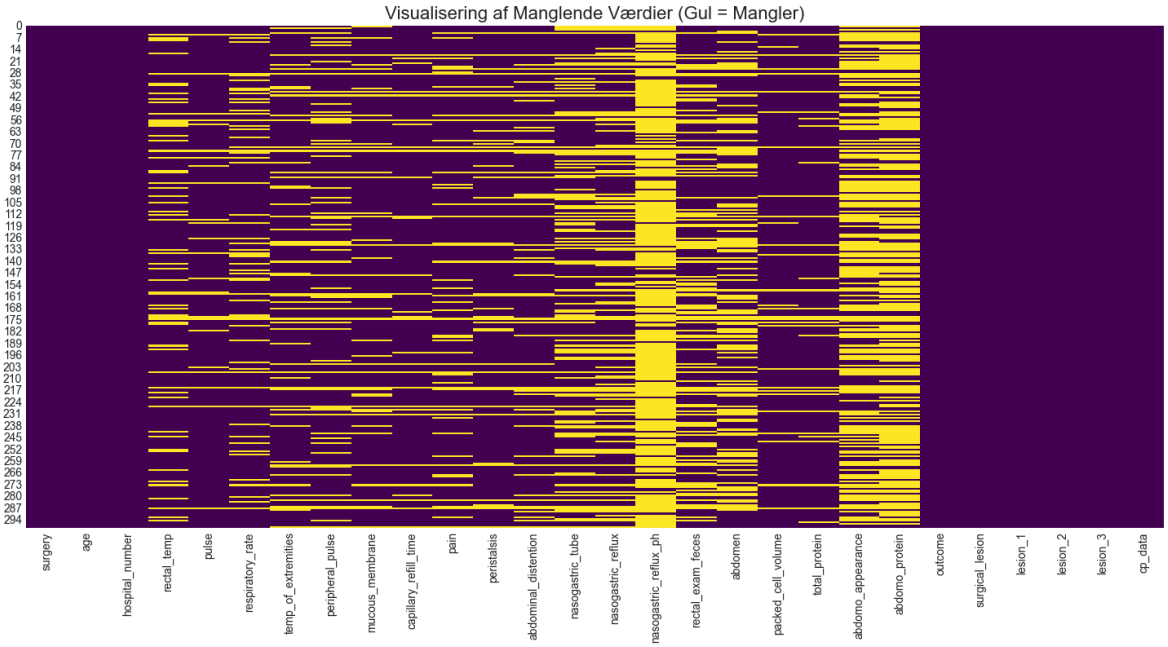
```
nasogastric_reflux       35.451505
nasogastric_tube         34.782609
rectal_exam_feces        34.113712
peripheral_pulse         23.076923
rectal_temp              20.066890
respiratory_rate         19.397993
temp_of_extremities      18.729097
abdominal_distention     18.729097
pain                     18.394649
mucous_membrane          15.719064
peristalsis              14.715719
total_protein            11.036789
capillary_refill_time    10.702341
packed_cell_volume        9.698997
pulse                     8.026756
dtype: float64
```

Visualisering af Manglende Data

Et heatmap er en effektiv måde visuelt at identificere mønstre i manglende data, som diskuteret i Lektion 5 om visualisering. Gule områder i heatmappet indikerer manglende værdier (NaN).

In [30]:
```
plt.figure(figsize=(18, 8))
sns.heatmap(df_original.isnull(), cbar=False, cmap='viridis')
plt.title('Visualisering af Manglende Værdier (Gul = Mangler)', fontsize=16)
plt.show()

print("\n Procentdel Manglende Værdier pr. Kolonne (sorteret)")
print(missing_percentage[missing_percentage > 0].sort_values(ascending=False))
```



```
 Procentdel Manglende Værdier pr. Kolonne (sorteret)
nasogastric_reflux_ph    82.274247
abdomo_protein           66.220736
abdomo_appearance        55.183946
abdomen                  39.464883
nasogastric_reflux       35.451505
nasogastric_tube         34.782609
rectal_exam_feces        34.113712
peripheral_pulse         23.076923
rectal_temp              20.066890
respiratory_rate         19.397993
temp_of_extremities      18.729097
abdominal_distention     18.729097
pain                     18.394649
mucous_membrane          15.719064
peristalsis              14.715719
total_protein            11.036789
capillary_refill_time    10.702341
packed_cell_volume        9.698997
pulse                     8.026756
dtype: float64
```

Trin 2.1: Diskussion af Typer af Datamangel (MCAR, MAR, NMAR)

Baseret på materialet fra Lektion 3, skal vi nu forsøge at identificere de sandsynlige typer af datamangel i vores datasæt. Dette er ofte en udfordring uden dybdegående domæneviden, men vi kan lave kvalificerede gæt baseret på mønstrene og kolonnebeskrivelser.

- **MCAR (Missing Completely At Random):** Sandsynligheden for at en værdi mangler er helt uafhængig af både observerede og uobserverede værdier. Dette kunne skyldes tilfældige tastefejl, midlertidige udstyrsfejl, eller at en prøve ved et uheld blev tabt. Små mængder manglende data i ellers velindsamlede kolonner *kan* være MCAR.
- **MAR (Missing At Random):** Sandsynligheden for at en værdi mangler afhænger *kun* af andre *observerede* værdier i datasættet, men ikke af den manglende værdi selv. F.eks. hvis et bestemt spørgsmål i en survey oftere springes over af en bestemt aldersgruppe (som er observeret).
- **NMAR (Not Missing At Random):** Sandsynligheden for at en værdi mangler afhænger af selve den manglende værdi (eller andre uobserverede faktorer). F.eks. hvis personer med meget høj indkomst er mindre tilbøjelige til at oplyse den. Dette er den sværeste type at håndtere korrekt.

Datasættet indeholder sandsynligvis en blanding af MAR og NMAR, især for kolonner med høj mangelrate. Der kan også være MCAR-elementer. Den høje grad af mangel i visse kolonner antyder, at simple imputationsmetoder (som gennemsnit/median) kan skabe betydelig bias, især hvis data er NMAR. For denne øvelse vil vi dog fortsætte med at inkludere de fleste kolonner for at demonstrere imputeringsteknikkerne, men i et reelt projekt ville en dybere analyse og evt. fjernelse af kolonner med ekstrem mangel være nødvendig.

Trin 2.2: Identifikation af Kolonnetyper

For at kunne anvende passende imputationsstrategier (f.eks. median for numeriske, modus for kategoriske), adskiller vi kolonnerne baseret på deres formodede datatype (`float64` / `int64` for numeriske, `object` for kategoriske).

In [31]:
```
df = df_original.copy()

cols_to_convert = ['packed_cell_volume', 'total_protein', 'abdomo_protein', 'nas

for col in cols_to_convert:
    df[col] = pd.to_numeric(df[col], errors='coerce')
    print(f"Kolonne '{col}' konverteret til {df[col].dtype}.")

numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
categorical_cols = df.select_dtypes(include='object').columns.tolist()

missing_cols = df.columns[df.isnull().any()].tolist()
numeric_cols_missing = [col for col in numeric_cols if col in missing_cols]
categorical_cols_missing = [col for col in categorical_cols if col in missing_co
```

```
Kolonne 'packed_cell_volume' konverteret til float64.
Kolonne 'total_protein' konverteret til float64.
Kolonne 'abdomo_protein' konverteret til float64.
Kolonne 'nasogastric_reflux_ph' konverteret til float64.
```

Trin 3: Data Preprocessing for Imputering og Modellering

Før vi kan anvende imputeringsteknikker (som KNNImputer) og træne klassifikationsmodeller, skal vi forberede data yderligere:

1. **Håndtering af kategoriske features:** Vi anvender `LabelEncoder` til at konvertere kategoriske strenge til heltal. Vi er opmærksomme på, at dette indfører en arbitrær orden, hvilket kan være problematisk for nogle modeller, men det er en simpel start. One-Hot Encoding ville være et alternativ.
2. **Definition af Features (X) og Target (y):** Vi vælger `outcome` som vores target-variabel (det vi vil forudsige). Resten af kolonnerne (undtagen irrelevante ID'er som `hospital_number`) bliver vores features (X).
3. **Opdeling i Trænings- og Testsæt:** Dette er *essentielt* og skal gøres *før* imputering. Vi bruger `train_test_split` til at opdele data, så vi kan træne vores imputere og modeller på træningssættet og evaluere dem på et uset testsæt. Dette forhindrer data leakage, hvor information fra testsættet utilsigtet påvirker træningen. Vi bruger `stratify=y` for at sikre, at fordelingen af `outcome`-klasserne er ens i trænings- og testsættet.

In [32]:
```
df_processed = df.copy()
label_encoders = {}

for col in categorical_cols:
    df_processed[col] = df_processed[col].fillna('Missing_Category').astype(str)
    le = LabelEncoder()
    le.fit(df_processed[col].unique())
    df_processed[col] = le.transform(df_processed[col])
    label_encoders[col] = le

df_processed.info()
```

```
target_col = 'outcome'
y = df_processed[target_col]
cols_to_drop = [target_col, 'hospital_number', 'lesion_1', 'lesion_2', 'lesion_3
X = df_processed.drop(columns=cols_to_drop)
feature_names = X.columns.tolist()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 28 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   surgery                299 non-null    int32
 1   age                    299 non-null    int32
 2   hospital_number        299 non-null    int64
 3   rectal_temp            239 non-null    float64
 4   pulse                  275 non-null    float64
 5   respiratory_rate       241 non-null    float64
 6   temp_of_extremities    299 non-null    int32
 7   peripheral_pulse       299 non-null    int32
 8   mucous_membrane        299 non-null    int32
 9   capillary_refill_time  299 non-null    int32
 10  pain                   299 non-null    int32
 11  peristalsis            299 non-null    int32
 12  abdominal_distention   299 non-null    int32
 13  nasogastric_tube       299 non-null    int32
 14  nasogastric_reflux     299 non-null    int32
 15  nasogastric_reflux_ph  53 non-null     float64
 16  rectal_exam_feces      299 non-null    int32
 17  abdomen                299 non-null    int32
 18  packed_cell_volume     270 non-null    float64
 19  total_protein          266 non-null    float64
 20  abdomo_appearance      299 non-null    int32
 21  abdomo_protein         101 non-null    float64
 22  outcome                299 non-null    int32
 23  surgical_lesion        299 non-null    int32
 24  lesion_1               299 non-null    int64
 25  lesion_2               299 non-null    int64
 26  lesion_3               299 non-null    int64
 27  cp_data                299 non-null    int32
dtypes: float64(7), int32(17), int64(4)
memory usage: 45.7 KB
X_train shape: (209, 23)
X_test shape: (90, 23)
y_train shape: (209,)
y_test shape: (90,)
```

Trin 4: Imputationsmetode 1: SimpleImputer (Median/Modus)

Vores første imputationsmetode er `SimpleImputer` fra `scikit-learn`. Vi vælger følgende strategier:

- **Median:** For numeriske kolonner. Medianen er mere robust over for outliers end middelværdien.
- **Modus (Most Frequent):** For de kolonner, der oprindeligt var kategoriske (nu label-encoded). Modus er den mest passende simple strategi for kategoriske data.

In [33]:
```
X_train_simple = X_train.copy()
X_test_simple = X_test.copy()

median_imputer = SimpleImputer(strategy='median')
median_imputer.fit(X_train_simple[numeric_cols_missing])
X_train_simple[numeric_cols_missing] = median_imputer.transform(X_train_simple[n
X_test_simple[numeric_cols_missing] = median_imputer.transform(X_test_simple[num

mode_imputer = SimpleImputer(strategy='most_frequent')
mode_imputer.fit(X_train_simple[categorical_cols_missing])
X_train_simple[categorical_cols_missing] = mode_imputer.transform(X_train_simple
X_test_simple[categorical_cols_missing] = mode_imputer.transform(X_test_simple[c

print(f"Manglende værdier i X_train_simple: {X_train_simple.isnull().sum().sum()
print(f"Manglende værdier i X_test_simple: {X_test_simple.isnull().sum().sum()}"
```

```
Manglende værdier i X_train_simple: 0
Manglende værdier i X_test_simple: 0
```

Trin 5: Imputationsmetode 2: KNNImputer

Vores anden metode er `KNNImputer`. Denne metode er mere avanceret end `SimpleImputer`, da den estimerer manglende værdier baseret på værdierne hos de nærmeste naboer i feature-rummet (K-Nearest Neighbors).

Vi bruger standardværdien `n_neighbors=5`.

In [34]:
```
X_train_knn = X_train.copy()
X_test_knn = X_test.copy()

knn_imputer = KNNImputer(n_neighbors=5)
X_train_knn = pd.DataFrame(knn_imputer.fit_transform(X_train_knn), columns=featu
X_test_knn = pd.DataFrame(knn_imputer.transform(X_test_knn), columns=feature_nam

print(f"Manglende værdier i X_train_knn: {X_train_knn.isnull().sum().sum()}")
print(f"Manglende værdier i X_test_knn: {X_test_knn.isnull().sum().sum()}")
```

```
Manglende værdier i X_train_knn: 0
Manglende værdier i X_test_knn: 0
```

Trin 6: Sammenligning af Imputeringsmetoder via Klassifikation

Nu hvor vi har to versioner af vores datasæt – en imputeret med `SimpleImputer` og en med `KNNImputer` – kan vi sammenligne effekten af imputeringen. En måde at gøre dette på er at træne en klassifikationsmodel på hvert af de imputerede træningssæt og evaluere dens performance på de tilsvarende testsæt. Vi bruger `KNeighborsClassifier` (igen med K=5) som vores model. Vi bruger nøjagtighed (accuracy) som vores evalueringsmetrik.

In [35]:
```
classifier = KNeighborsClassifier(n_neighbors=5)
```

```
classifier.fit(X_train_simple, y_train)
y_pred_simple = classifier.predict(X_test_simple)
accuracy_simple = accuracy_score(y_test, y_pred_simple)

classifier.fit(X_train_knn, y_train)
y_pred_knn = classifier.predict(X_test_knn)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

print(f"Accuracy (SimpleImputer): {accuracy_simple:.4f}")
print(f"Accuracy (KNNImputer):    {accuracy_knn:.4f}")
```

```
Accuracy (SimpleImputer): 0.6222
Accuracy (KNNImputer):    0.6556
```

Trin 7: Strukturering af Data i Relationelle Databaser

Vi vælger at bruge SQLite, da det er en simpel process, der er indbygget i Python via `sqlite3` modulet.

Vores Databasestruktur: Vi opretter tre tabeller for at demonstrere relationer:

1. `Horses` : Indeholder grundlæggende information om hver hest, identificeret ved `hospital_number` (primærnøgle).
2. `MedicalObservations` : Indeholder de kliniske målinger og observationer for hver hest. Den har sin egen primærnøgle (`observation_id`) og en fremmednøgle (`hospital_number`), der linker til `Horses` -tabellen.
3. `LesionInfo` : Indeholder information om eventuelle læsioner (`lesion_1` , `lesion_2` , `lesion_3`). Den har også en fremmednøgle (`hospital_number`) til `Horses` .

Vi bruger det *originale* DataFrame (`df_original`) til at populere databasen for at bevare de oprindelige værdier, inklusiv `NaN` , som SQLite håndterer som `NULL` .

In [36]:
```
db_file = 'horse_data_workshop.db'
conn = sqlite3.connect(db_file)
cursor = conn.cursor()

cursor.execute("DROP TABLE IF EXISTS LesionInfo")
cursor.execute("DROP TABLE IF EXISTS MedicalObservations")
cursor.execute("DROP TABLE IF EXISTS Horses")

cursor.execute("""
    CREATE TABLE Horses (
        hospital_number INTEGER PRIMARY KEY,
        surgery TEXT,
        age TEXT,
        rectal_temp REAL,
        pulse REAL
    );
""")

cursor.execute("""
    CREATE TABLE MedicalObservations (
        observation_id INTEGER PRIMARY KEY AUTOINCREMENT,
        hospital_number INTEGER NOT NULL,
        respiratory_rate REAL,
        temp_of_extremities TEXT,
```

```
        peripheral_pulse TEXT,
        mucous_membrane TEXT,
        capillary_refill_time TEXT,
        pain TEXT,
        peristalsis TEXT,
        abdominal_distention TEXT,
        nasogastric_tube TEXT,
        nasogastric_reflux TEXT,
        nasogastric_reflux_ph REAL,
        rectal_exam_feces TEXT,
        abdomen TEXT,
        packed_cell_volume REAL,
        total_protein REAL,
        abdomo_appearance TEXT,
        abdomo_protein REAL,
        surgical_lesion TEXT,
        cp_data TEXT,
        outcome TEXT,
        FOREIGN KEY (hospital_number) REFERENCES Horses(hospital_number)
    );
""")

cursor.execute("""
    CREATE TABLE LesionInfo (
        lesion_id INTEGER PRIMARY KEY AUTOINCREMENT,
        hospital_number INTEGER NOT NULL,
        lesion_1 INTEGER,
        lesion_2 INTEGER,
        lesion_3 INTEGER,
        FOREIGN KEY (hospital_number) REFERENCES Horses(hospital_number)
    );
""")

horses_df = df_original[['hospital_number', 'surgery', 'age', 'rectal_temp', 'pu
horses_df.to_sql('Horses', conn, if_exists='append', index=False)

obs_cols = ['hospital_number', 'respiratory_rate', 'temp_of_extremities', 'perip
            'mucous_membrane', 'capillary_refill_time', 'pain', 'peristalsis',
            'abdominal_distention', 'nasogastric_tube', 'nasogastric_reflux',
            'nasogastric_reflux_ph', 'rectal_exam_feces', 'abdomen',
            'packed_cell_volume', 'total_protein', 'abdomo_appearance',
            'abdomo_protein', 'surgical_lesion', 'cp_data', 'outcome']
med_obs_df = df[obs_cols].where(pd.notnull(df[obs_cols]), None)
med_obs_df.to_sql('MedicalObservations', conn, if_exists='append', index=False)

lesion_cols = ['hospital_number', 'lesion_1', 'lesion_2', 'lesion_3']
lesion_df = df_original[lesion_cols].where(pd.notnull(df_original[lesion_cols]),
lesion_df.to_sql('LesionInfo', conn, if_exists='append', index=False)

conn.commit()
conn.close()
```

Trin 8: Konklusion og Refleksion

I Trin 6 sammenlignede vi de to imputeringsmetoder ved at træne en KNeighborsClassifier (med k=5) på de to resulterende datasæt (X_train_simple, X_train_knn) og evaluere dem på de tilsvarende testsæt (X_test_simple, X_test_knn). Vi observerede følgende nøjagtigheder (accuracy):

Accuracy med SimpleImputer: 0.6222
Accuracy med KNNImputer: 0.6556

Konklusionen var, at KNNImputer gav en højere nøjagtighed end SimpleImputer i dette specifikke setup, hvilket potentielt skyldes dens evne til at udnytte relationer mellem features.