Data Analysis

We will work with model Diabetes130US from openml with id 45069, first we will load this dataset.

```
import openml
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import torch


dataset_id = 45069

dataset = openml.datasets.get_dataset(dataset_id)
X, y, _, _ = dataset.get_data(dataset_format="dataframe",
target=dataset.default_target_attribute)


if y is not None:
    df = pd.concat([X, y], axis=1)
else:
    df = X

df.head()
```

```
   admission_type_id  discharge_disposition_id  admission_source_id  \
0                  1                         3                    7
1                  5                         1                    1
2                  2                         1                    1
3                  3                         1                    1
4                  3                         1                    1

   time_in_hospital  num_lab_procedures  num_procedures
num_medications  \
0                 4                  41               2
17
1                 3                  64               0
10
2                 2                  39               0
8
3                 1                  39               2
12
4                 3                  21               2
23

   number_outpatient  number_emergency  number_inpatient  ...
```

```
citoglipton  \
0                   0                   0                   0  ...
No
1                   0                   0                   1  ...
No
2                   0                   0                   0  ...
No
3                   0                   0                   1  ...
No
4                   1                   0                   0  ...
No

  insulin glyburide.metformin glipizide.metformin
glimepiride.pioglitazone  \
0  Steady                  No                  No
No
1  Steady                  No                  No
No
2  Steady                  No                  No
No
3      No                  No                  No
No
4      No                  No                  No
No

  metformin.rosiglitazone metformin.pioglitazone change diabetesMed
class
0                      No                      No     Ch          Yes
NO
1                      No                      No     Ch          Yes
NO
2                      No                      No     Ch          Yes
NO
3                      No                      No     No           No
NO
4                      No                      No     Ch          Yes
NO

[5 rows x 48 columns]
```

First we load the dataset from OpenML (if we run this for the first time we also download OpenML with pip).

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 48 columns):
 #   Column                    Non-Null Count   Dtype
```

```
---  ------                      --------------   -----
 0   admission_type_id           101766 non-null  uint8
 1   discharge_disposition_id    101766 non-null  uint8
 2   admission_source_id         101766 non-null  uint8
 3   time_in_hospital            101766 non-null  uint8
 4   num_lab_procedures          101766 non-null  uint8
 5   num_procedures              101766 non-null  uint8
 6   num_medications             101766 non-null  uint8
 7   number_outpatient           101766 non-null  uint8
 8   number_emergency            101766 non-null  uint8
 9   number_inpatient            101766 non-null  uint8
 10  number_diagnoses            101766 non-null  uint8
 11  race                        99493 non-null   category
 12  gender                      101766 non-null  category
 13  age                         101766 non-null  category
 14  weight                      3197 non-null    category
 15  payer_code                  61510 non-null   category
 16  medical_specialty           51817 non-null   category
 17  diag_1                      101745 non-null  category
 18  diag_2                      101408 non-null  category
 19  diag_3                      100343 non-null  category
 20  max_glu_serum               101766 non-null  category
 21  A1Cresult                   101766 non-null  category
 22  metformin                   101766 non-null  category
 23  repaglinide                 101766 non-null  category
 24  nateglinide                 101766 non-null  category
 25  chlorpropamide              101766 non-null  category
 26  glimepiride                 101766 non-null  category
 27  acetohexamide               101766 non-null  category
 28  glipizide                   101766 non-null  category
 29  glyburide                   101766 non-null  category
 30  tolbutamide                 101766 non-null  category
 31  pioglitazone                101766 non-null  category
 32  rosiglitazone               101766 non-null  category
 33  acarbose                    101766 non-null  category
 34  miglitol                    101766 non-null  category
 35  troglitazone                101766 non-null  category
 36  tolazamide                  101766 non-null  category
 37  examide                     101766 non-null  category
 38  citoglipton                 101766 non-null  category
 39  insulin                     101766 non-null  category
 40  glyburide.metformin         101766 non-null  category
 41  glipizide.metformin         101766 non-null  category
 42  glimepiride.pioglitazone    101766 non-null  category
 43  metformin.rosiglitazone     101766 non-null  category
 44  metformin.pioglitazone      101766 non-null  category
 45  change                      101766 non-null  category
 46  diabetesMed                 101766 non-null  category
 47  class                       101766 non-null  object
```

```
dtypes: category(36), object(1), uint8(11)
memory usage: 5.7+ MB
```

We can see that we have 101 766 records with 48 features, 36 being categorical, 1 being object and 11 being integer.

```
df.nunique()
```

```
admission_type_id             8
discharge_disposition_id     26
admission_source_id          17
time_in_hospital             14
num_lab_procedures          118
num_procedures                7
num_medications              75
number_outpatient            39
number_emergency             33
number_inpatient             21
number_diagnoses             16
race                          5
gender                        3
age                          10
weight                        9
payer_code                   17
medical_specialty            72
diag_1                      716
diag_2                      748
diag_3                      789
max_glu_serum                 4
A1Cresult                     4
metformin                     4
repaglinide                   4
nateglinide                   4
chlorpropamide                4
glimepiride                   4
acetohexamide                 2
glipizide                     4
glyburide                     4
tolbutamide                   2
pioglitazone                  4
rosiglitazone                 4
acarbose                      4
miglitol                      4
troglitazone                  2
tolazamide                    3
examide                       1
citoglipton                   1
insulin                       4
glyburide.metformin           4
```

```
glipizide.metformin           2
glimepiride.pioglitazone      2
metformin.rosiglitazone       2
metformin.pioglitazone        2
change                        2
diabetesMed                   2
class                         3
dtype: int64
```

We can see that examine and citoglipton have only 1 unique value, therefore we can drop them. We will also drop weight feature since it is filled in only 3197 times (which is about 3.14%).

```
columns_to_drop = ['examide', 'citoglipton', 'weight']

df = df.drop(columns=columns_to_drop)

print(f"Columns {columns_to_drop} have been dropped.")
print("\nDataFrame columns after dropping:")
print(df.columns)
```

```
Columns ['examide', 'citoglipton', 'weight'] have been dropped.

DataFrame columns after dropping:
Index(['admission_type_id', 'discharge_disposition_id',
'admission_source_id',
       'time_in_hospital', 'num_lab_procedures', 'num_procedures',
       'num_medications', 'number_outpatient', 'number_emergency',
       'number_inpatient', 'number_diagnoses', 'race', 'gender',
'age',
       'payer_code', 'medical_specialty', 'diag_1', 'diag_2',
'diag_3',
       'max_glu_serum', 'A1Cresult', 'metformin', 'repaglinide',
'nateglinide',
       'chlorpropamide', 'glimepiride', 'acetohexamide', 'glipizide',
       'glyburide', 'tolbutamide', 'pioglitazone', 'rosiglitazone',
'acarbose',
       'miglitol', 'troglitazone', 'tolazamide', 'insulin',
       'glyburide.metformin', 'glipizide.metformin',
       'glimepiride.pioglitazone', 'metformin.rosiglitazone',
       'metformin.pioglitazone', 'change', 'diabetesMed', 'class'],
      dtype='object')
```

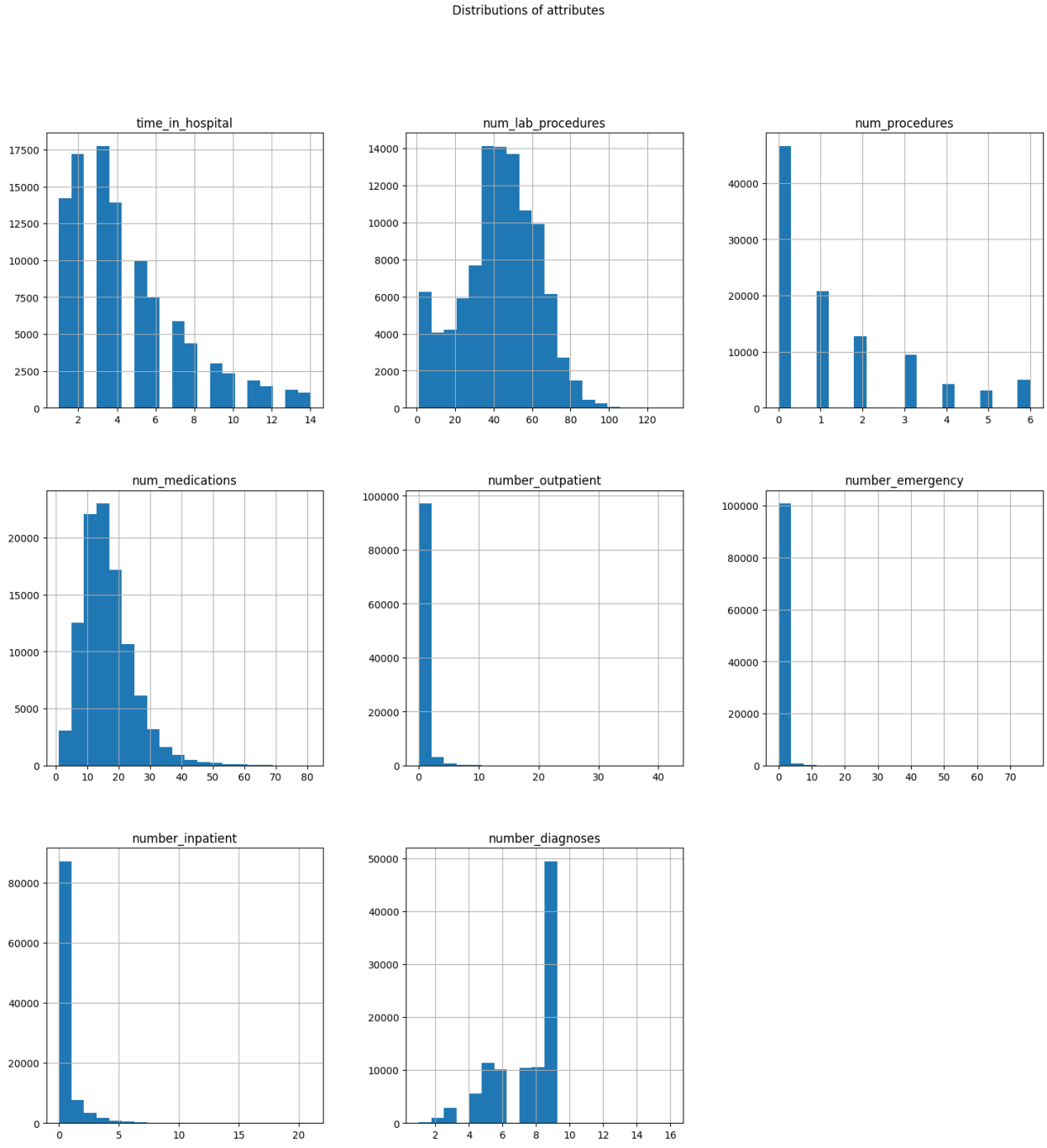
Next we will take features that represent ids and feature class and make it into categorical features.

```
columns_to_convert_to_category = ['admission_type_id',
'discharge_disposition_id', 'admission_source_id', 'class']

for col in columns_to_convert_to_category:
```

```
    df[col] = df[col].astype('category')

print("Converted columns to category:")
print(df[columns_to_convert_to_category].dtypes)
```

```
Converted columns to category:
admission_type_id           category
discharge_disposition_id    category
admission_source_id         category
class                       category
dtype: object
```

```
df.hist(figsize=(18, 18), bins=20)
plt.suptitle("Distributions of attributes")
plt.show()
```




```
categorical_cols = df.select_dtypes(include='category').columns

print("Unique values for categorical features:")

for col in categorical_cols:
    print(f"\nColumn: {col}")
    print(df[col].unique())
```

```
Unique values for categorical features:

Column: admission_type_id
[1, 5, 2, 3, 6, 8, 4, 7]
Categories (8, uint8): [1, 2, 3, 4, 5, 6, 7, 8]

Column: discharge_disposition_id
[3, 1, 6, 22, 18, ..., 10, 17, 27, 19, 20]
Length: 26
Categories (26, uint8): [1, 2, 3, 4, ..., 24, 25, 27, 28]

Column: admission_source_id
[7, 1, 17, 6, 4, ..., 8, 22, 11, 25, 13]
Length: 17
Categories (17, uint8): [1, 2, 3, 4, ..., 17, 20, 22, 25]

Column: race
['AfricanAmerican', 'Caucasian', 'Hispanic', 'Other', NaN, 'Asian']
Categories (5, object): ['AfricanAmerican' < 'Asian' < 'Caucasian' <
'Hispanic' < 'Other']

Column: gender
['Female', 'Male', 'Unknown/Invalid']
Categories (3, object): ['Female' < 'Male' < 'Unknown/Invalid']

Column: age
['[60-70)', '[50-60)', '[80-90)', '[40-50)', '[70-80)', '[90-100)',
'[10-20)', '[30-40)', '[20-30)', '[0-10)']
Categories (10, object): ['[0-10)' < '[10-20)' < '[20-30)' < '[30-40)'
... '[60-70)' < '[70-80)' < '[80-90)' < '[90-100)']

Column: payer_code
['MC', 'MD', NaN, 'BC', 'SP', ..., 'DM', 'WC', 'SI', 'OT', 'FR']
Length: 18
Categories (17, object): ['BC' < 'CH' < 'CM' < 'CP' ... 'SI' < 'SP' <
'UN' < 'WC']

Column: medical_specialty
[NaN, 'Surgery-Vascular', 'InternalMedicine', 'Surgery-General',
'Family/GeneralPractice', ..., 'Surgery-PlasticwithinHeadandNeck',
'Dermatology', 'Perinatology', 'Pediatrics-InfectiousDiseases',
'Psychiatry-Addictive']
Length: 73
Categories (72, object): ['AllergyandImmunology' < 'Anesthesiology' <
'Anesthesiology-Pediatric' < 'Cardiology' ... 'Surgery-Thoracic' <
'Surgery-Vascular' < 'SurgicalSpecialty' < 'Urology']

Column: diag_1
['820', '276', '493', '250.4', '218', ..., '133', '913', '827', '373',
'837']
```

```
Length: 717
Categories (716, object): ['10' < '11' < '110' < '112' ... 'V66' <
'V67' < 'V70' < 'V71']

Column: diag_2
['599', '295', '250', '403', '627', ..., '915', '268', '665', '908',
'894']
Length: 749
Categories (748, object): ['11' < '110' < '111' < '112' ... 'V70' <
'V72' < 'V85' < 'V86']

Column: diag_3
['294', '780', NaN, '285', '620', ..., '195', '684', 'E864', '111',
'838']
Length: 790
Categories (789, object): ['11' < '110' < '111' < '112' ... 'V70' <
'V72' < 'V85' < 'V86']

Column: max_glu_serum
['None', 'Norm', '>300', '>200']
Categories (4, object): ['>200' < '>300' < 'None' < 'Norm']

Column: A1Cresult
['None', '>8', '>7', 'Norm']
Categories (4, object): ['>7' < '>8' < 'None' < 'Norm']

Column: metformin
['No', 'Steady', 'Down', 'Up']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: repaglinide
['No', 'Steady', 'Up', 'Down']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: nateglinide
['No', 'Steady', 'Down', 'Up']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: chlorpropamide
['No', 'Steady', 'Up', 'Down']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: glimepiride
['No', 'Steady', 'Down', 'Up']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: acetohexamide
['No', 'Steady']
Categories (2, object): ['No' < 'Steady']

Column: glipizide
```

```
['Steady', 'No', 'Up', 'Down']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: glyburide
['No', 'Steady', 'Up', 'Down']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: tolbutamide
['No', 'Steady']
Categories (2, object): ['No' < 'Steady']

Column: pioglitazone
['No', 'Steady', 'Up', 'Down']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: rosiglitazone
['No', 'Steady', 'Down', 'Up']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: acarbose
['No', 'Steady', 'Down', 'Up']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: miglitol
['No', 'Steady', 'Up', 'Down']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: troglitazone
['No', 'Steady']
Categories (2, object): ['No' < 'Steady']

Column: tolazamide
['No', 'Steady', 'Up']
Categories (3, object): ['No' < 'Steady' < 'Up']

Column: insulin
['Steady', 'No', 'Down', 'Up']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: glyburide.metformin
['No', 'Steady', 'Down', 'Up']
Categories (4, object): ['Down' < 'No' < 'Steady' < 'Up']

Column: glipizide.metformin
['No', 'Steady']
Categories (2, object): ['No' < 'Steady']

Column: glimepiride.pioglitazone
['No', 'Steady']
Categories (2, object): ['No' < 'Steady']
```

```
Column: metformin.rosiglitazone
['No', 'Steady']
Categories (2, object): ['No' < 'Steady']

Column: metformin.pioglitazone
['No', 'Steady']
Categories (2, object): ['No' < 'Steady']

Column: change
['Ch', 'No']
Categories (2, object): ['Ch' < 'No']

Column: diabetesMed
['Yes', 'No']
Categories (2, object): ['No' < 'Yes']

Column: class
['NO', '>30', '<30']
Categories (3, object): ['<30', '>30', 'NO']
```

Next we drop duplicates.

```
print("Shape before removing duplicates:", df.shape)

df.drop_duplicates(inplace=True)

print("Shape after removing duplicates:", df.shape)
```

```
Shape before removing duplicates: (101766, 45)
Shape after removing duplicates: (101766, 45)
```

We have no duplicates.

```
print(df.isnull().sum())
```

```
admission_type_id               0
discharge_disposition_id        0
admission_source_id             0
time_in_hospital                0
num_lab_procedures              0
num_procedures                  0
num_medications                 0
number_outpatient               0
number_emergency                0
number_inpatient                0
number_diagnoses                0
race                         2273
gender                          0
age                             0
```

```
payer_code                  40256
medical_specialty           49949
diag_1                         21
diag_2                        358
diag_3                       1423
max_glu_serum                   0
A1Cresult                       0
metformin                       0
repaglinide                     0
nateglinide                     0
chlorpropamide                  0
glimepiride                     0
acetohexamide                   0
glipizide                       0
glyburide                       0
tolbutamide                     0
pioglitazone                    0
rosiglitazone                   0
acarbose                        0
miglitol                        0
troglitazone                    0
tolazamide                      0
insulin                         0
glyburide.metformin             0
glipizide.metformin             0
glimepiride.pioglitazone        0
metformin.rosiglitazone         0
metformin.pioglitazone          0
change                          0
diabetesMed                     0
class                           0
dtype: int64
```

We have null values in race, payer_code, medical_specialty, diag_1, diag_2, diag_3 columns, we will add missing value for race, payer_code and medical_specialty. For diagnoses we will make columns for each possible diagnosis value and mark it as 1 if its pressent in either of diag_1, diag_2 or diag_3, since it doesnt matter if the diagnosis is in diag_1 or diag_3.

```
categorical_cols_with_missing = ['race','payer_code',
'medical_specialty']


for col in categorical_cols_with_missing:
    if col in df.columns:
        if pd.api.types.is_categorical_dtype(df[col].dtype):
            if 'Missing' not in df[col].cat.categories:
                df[col] = df[col].cat.add_categories('Missing')
        df[col].fillna('Missing', inplace=True)
```

```
print("Missing values:")
display(df[categorical_cols_with_missing].isnull().sum())
```

```
Missing values:

C:\Users\maria\AppData\Local\Temp\ipykernel_26612\456492141.py:6:
DeprecationWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype,
pd.CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(df[col].dtype):
C:\Users\maria\AppData\Local\Temp\ipykernel_26612\456492141.py:9:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df[col].fillna('Missing', inplace=True)
```

```
race                 0
payer_code           0
medical_specialty    0
dtype: int64
```

```
all_diag_codes = pd.concat([df['diag_1'], df['diag_2'],
df['diag_3']]).unique()

all_diag_codes = [code for code in all_diag_codes if pd.notna(code)]

print(f"Number of unique diagnosis: {len(all_diag_codes)}")

print(all_diag_codes[:10])
```

```
Number of unique diagnosis: 915
['820', '276', '493', '250.4', '218', '250.6', '410', '486', '584',
'428']
```

```
diag_cols_dict = {
    f'diag|{code}': ((df['diag_1'] == code) |
                     (df['diag_2'] == code) |
                     (df['diag_3'] == code)).astype(int)
    for code in all_diag_codes
}
```

```
diag_cols_df = pd.DataFrame(diag_cols_dict, index=df.index)

df = df.drop(columns=['diag_1', 'diag_2', 'diag_3'])

df = pd.concat([df, diag_cols_df], axis=1)

print("New DataFrame:")
display(df.head())
```

```
New DataFrame:
```

```
  admission_type_id discharge_disposition_id admission_source_id  \
0                 1                        3                   7
1                 5                        1                   1
2                 2                        1                   1
3                 3                        1                   1
4                 3                        1                   1

   time_in_hospital  num_lab_procedures  num_procedures
num_medications  \
0                 4                  41               2
17
1                 3                  64               0
10
2                 2                  39               0
8
3                 1                  39               2
12
4                 3                  21               2
23

   number_outpatient  number_emergency  number_inpatient  ...  diag|
744  \
0                  0                 0                 0  ...
0
1                  0                 0                 1  ...
0
2                  0                 0                 0  ...
0
3                  0                 0                 1  ...
0
4                  1                 0                 0  ...
0

  diag|V22 diag|265 diag|E912 diag|E922 diag|E861 diag|E865 diag|
387  \
0        0        0         0         0         0         0        0

1        0        0         0         0         0         0        0
```

```
2         0         0          0          0          0          0          0

3         0         0          0          0          0          0          0

4         0         0          0          0          0          0          0


  diag|E966 diag|E864
0          0         0
1          0         0
2          0         0
3          0         0
4          0         0

[5 rows x 957 columns]
```

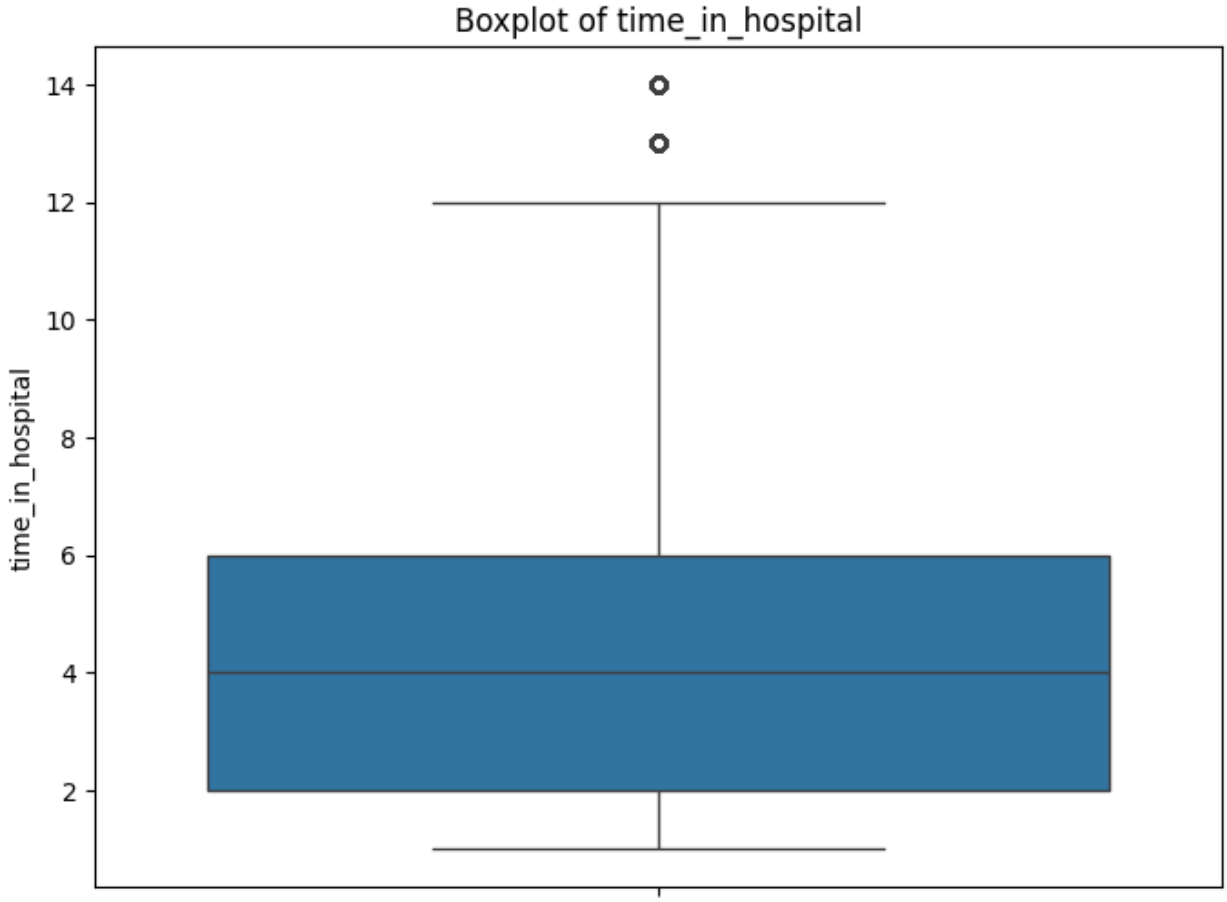
Next we will take a look at numerical features.

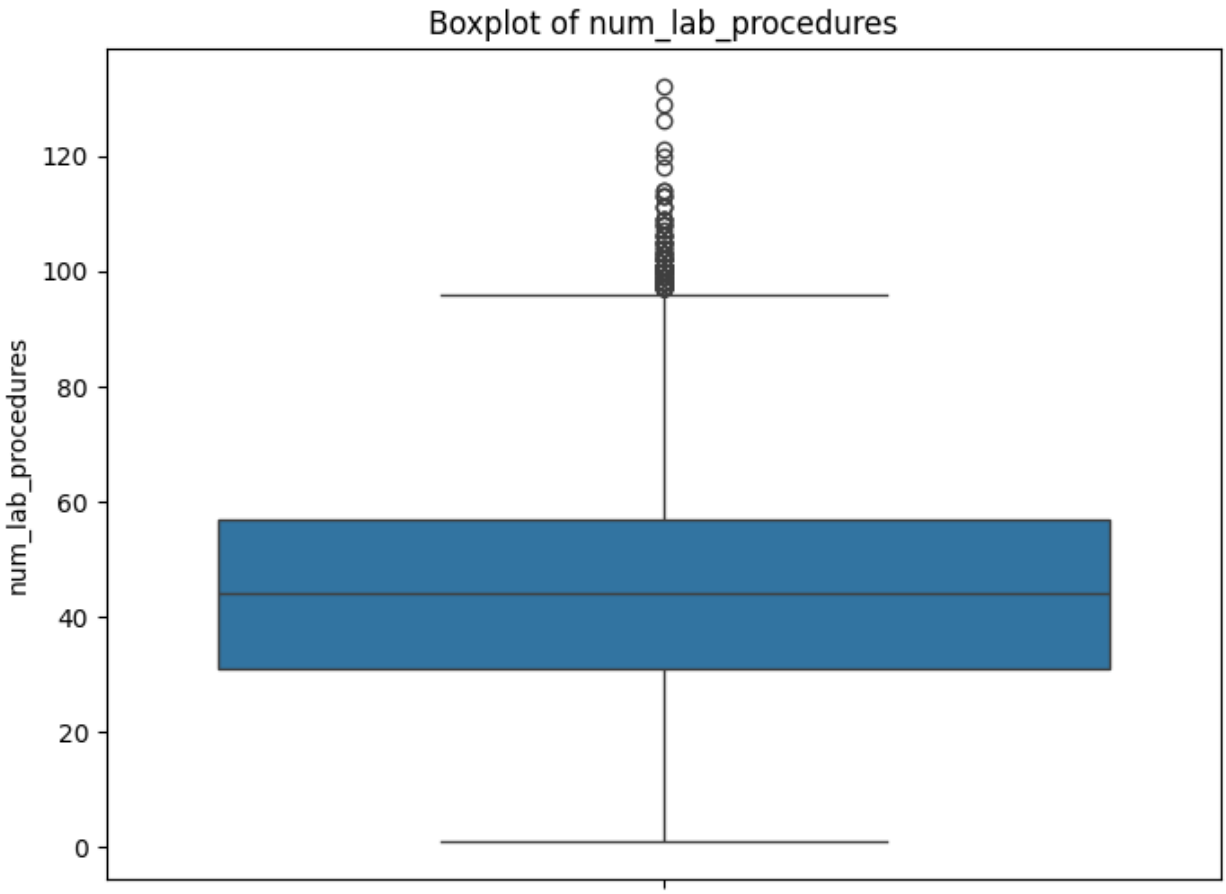
```
df.describe()

       time_in_hospital  num_lab_procedures  num_procedures
num_medications  \
count     101766.000000       101766.000000   101766.000000
101766.000000
mean           4.395987           43.095641        1.339730
16.021844
std            2.985108           19.674362        1.705807
8.127566
min            1.000000            1.000000        0.000000
1.000000
25%            2.000000           31.000000        0.000000
10.000000
50%            4.000000           44.000000        1.000000
15.000000
75%            6.000000           57.000000        2.000000
20.000000
max           14.000000          132.000000        6.000000
81.000000

       number_outpatient  number_emergency  number_inpatient  \
count      101766.000000     101766.000000     101766.000000
mean            0.369357          0.197836          0.635566
std             1.267265          0.930472          1.262863
min             0.000000          0.000000          0.000000
25%             0.000000          0.000000          0.000000
50%             0.000000          0.000000          0.000000
75%             0.000000          0.000000          1.000000
max            42.000000         76.000000         21.000000

       number_diagnoses       diag|820       diag|276  ...       diag|
```

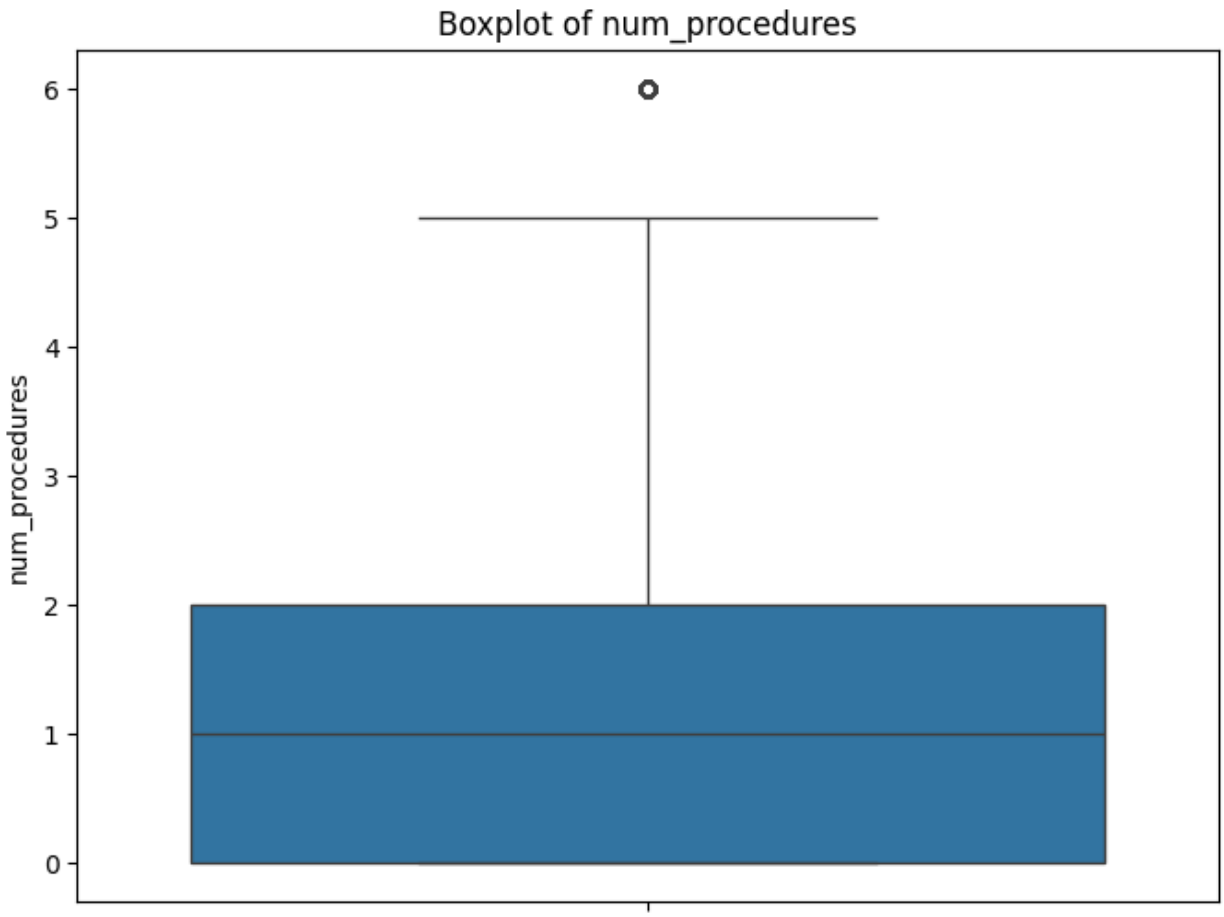
```
744  \
count     101766.000000  101766.000000  101766.000000  ...
101766.000000
mean           7.422607       0.011212       0.127361  ...
0.000010
std            1.933600       0.105292       0.333378  ...
0.003135
min            1.000000       0.000000       0.000000  ...
0.000000
25%            6.000000       0.000000       0.000000  ...
0.000000
50%            8.000000       0.000000       0.000000  ...
0.000000
75%            9.000000       0.000000       0.000000  ...
0.000000
max           16.000000       1.000000       1.000000  ...
1.000000

            diag|V22       diag|265      diag|E912      diag|E922  \
count  101766.000000  101766.000000  101766.000000  101766.000000
mean        0.000010       0.000010       0.000010       0.000010
std         0.003135       0.003135       0.003135       0.003135
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000

           diag|E861      diag|E865       diag|387      diag|E966  \
count  101766.000000  101766.000000  101766.000000  101766.000000
mean        0.000010       0.000010       0.000010       0.000010
std         0.003135       0.003135       0.003135       0.003135
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       0.000000
75%         0.000000       0.000000       0.000000       0.000000
max         1.000000       1.000000       1.000000       1.000000

           diag|E864
count  101766.000000
mean        0.000010
std         0.003135
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 923 columns]
```

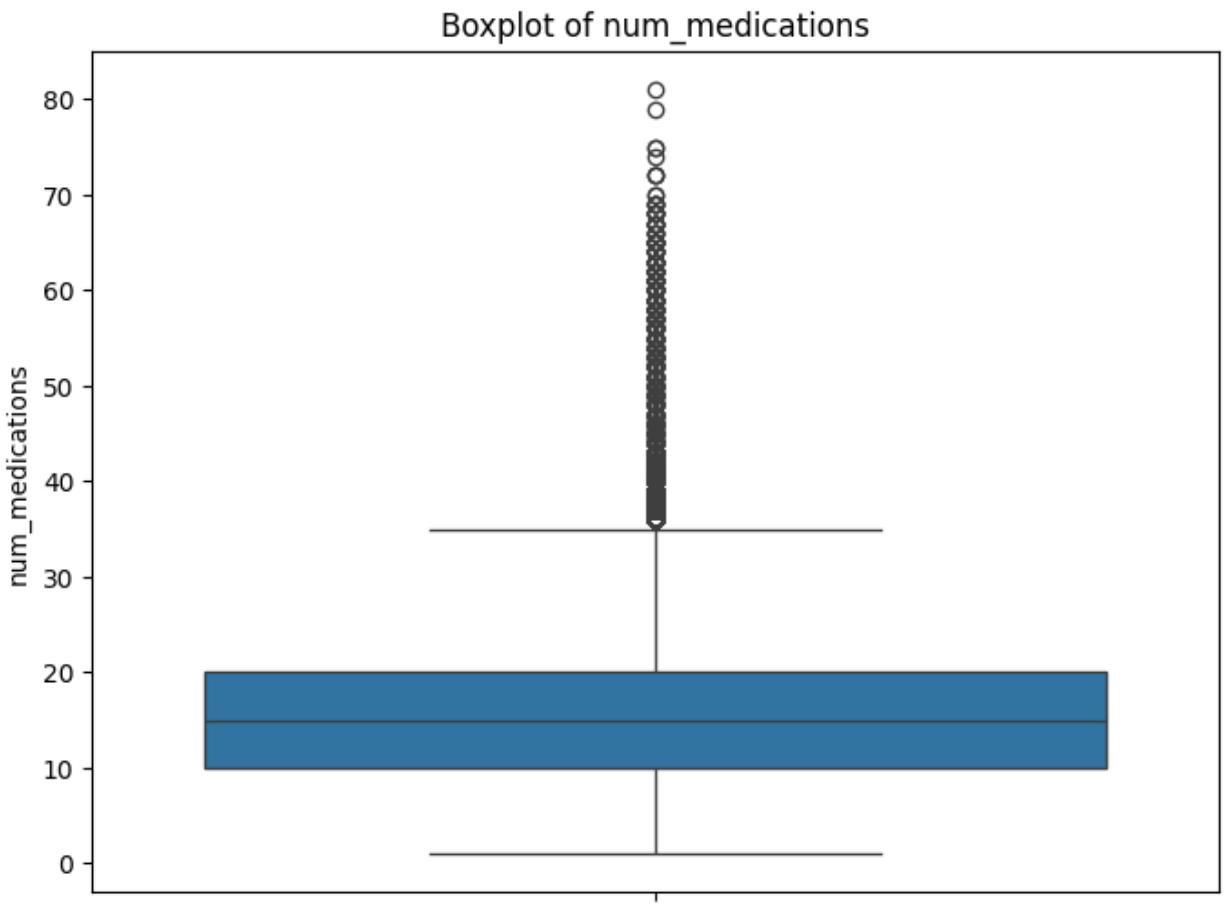
```
numerical_cols = ['admission_type_id', 'discharge_disposition_id',
'admission_source_id', 'time_in_hospital', 'num_lab_procedures',
'num_procedures', 'num_medications', 'number_outpatient',
'number_emergency', 'number_inpatient', 'number_diagnoses']
id_cols = ['admission_type_id', 'discharge_disposition_id',
'admission_source_id']

numerical_cols_for_plotting = [col for col in numerical_cols if col
not in id_cols]

print("Boxplots of selected numerical columns (excluding IDs):")
for col in numerical_cols_for_plotting:
    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df, y=col)
    plt.title(f'Boxplot of {col}')
    plt.ylabel(col)
    plt.show()
```

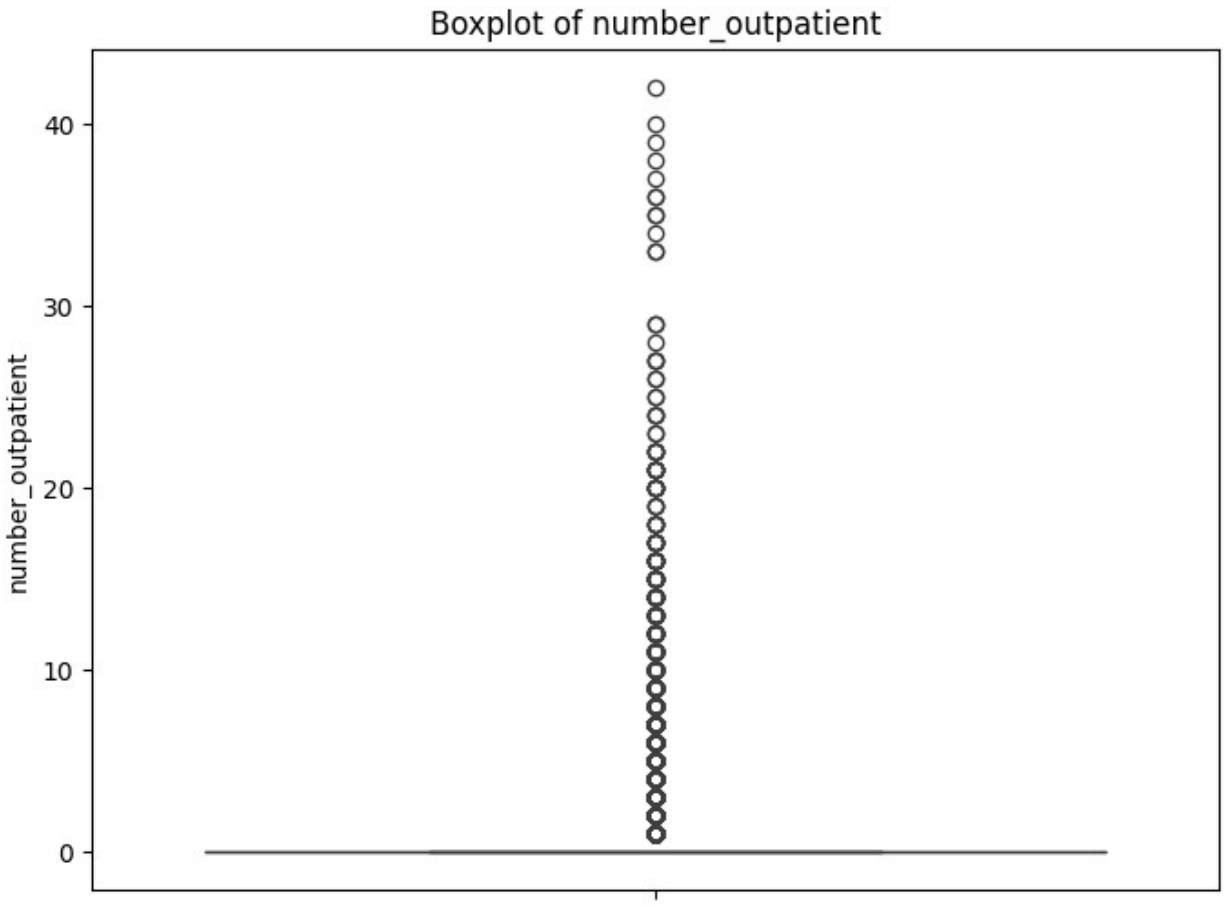
```
Boxplots of selected numerical columns (excluding IDs):
```

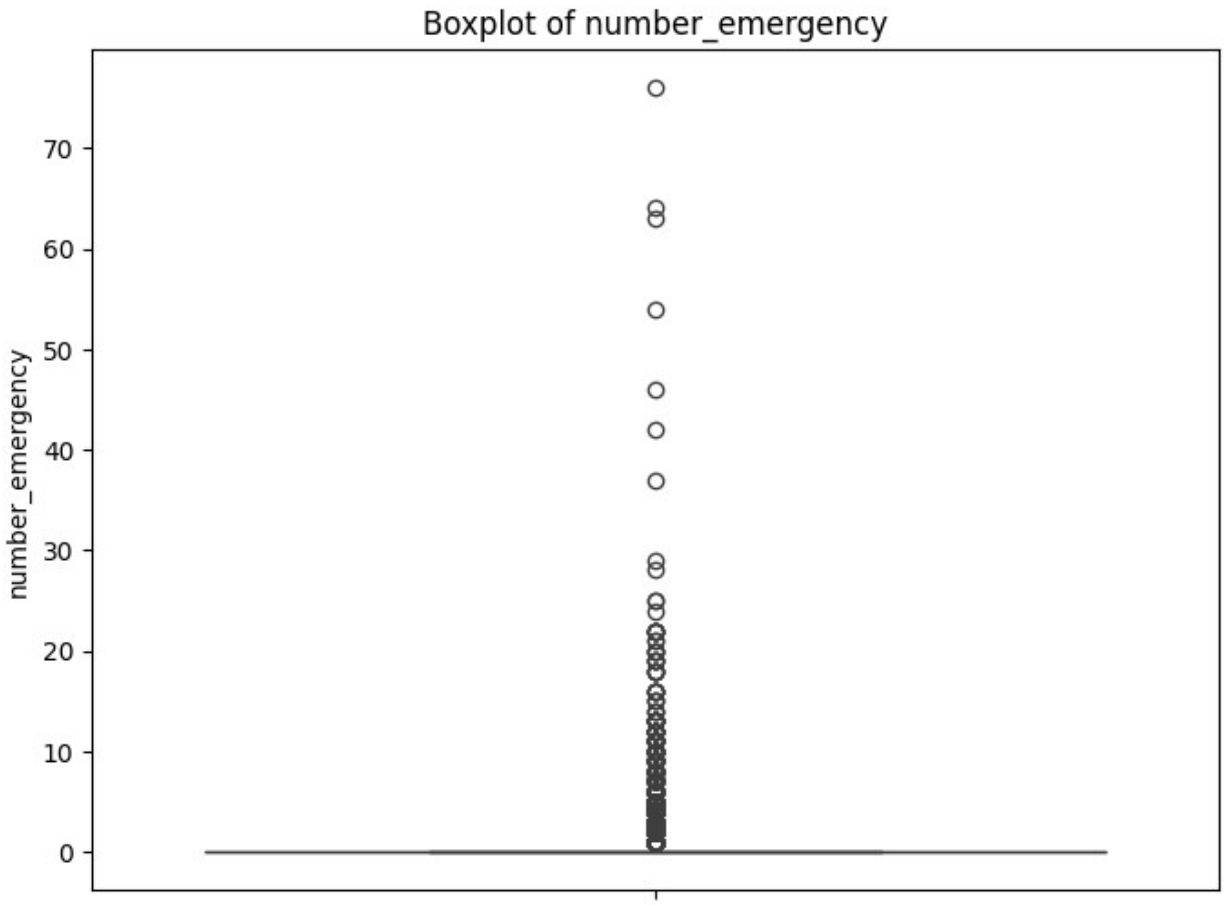


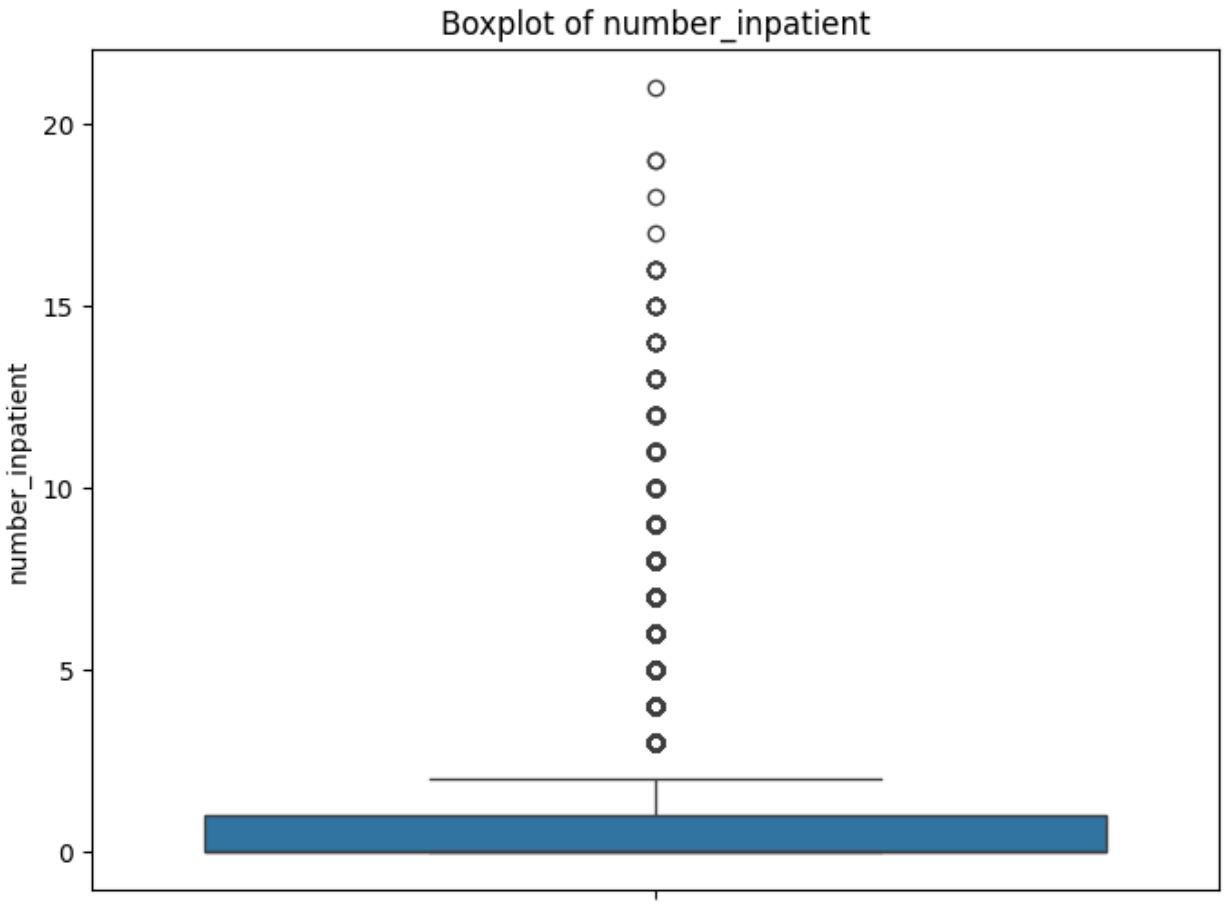

Boxplot of num_lab_procedures
num_lab_procedures
120
100
80
60
40
20
0

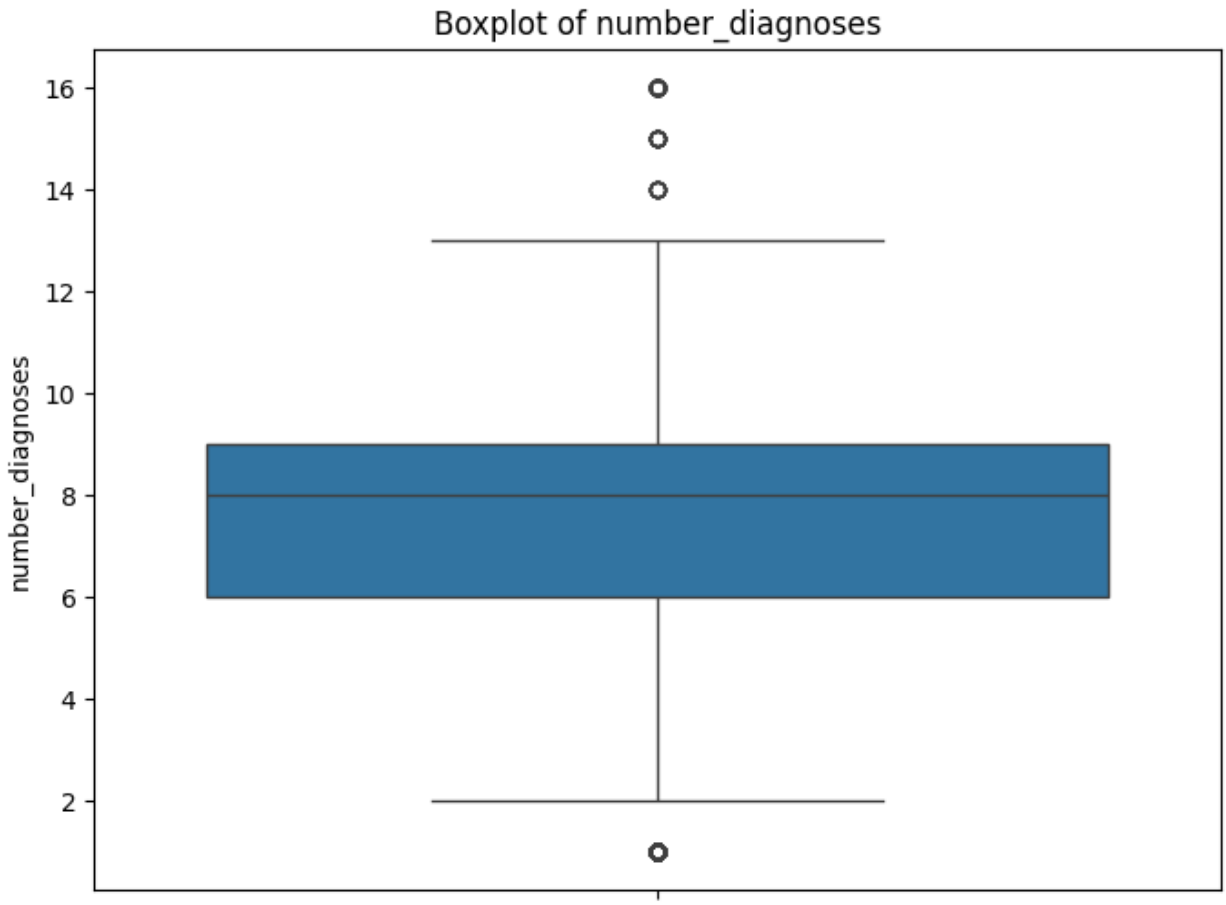

Boxplot of num_procedures
num_procedures
0
1
2
3
4
5
6


Boxplot of num_medications
num_medications
0
10
20
30
40
50
60
70
80


Boxplot of number_outpatient
number_outpatient
40
30
20
10
0


Boxplot of number_emergency
number_emergency
70
60
50
40
30
20
10
0


Boxplot of number_inpatient
number_inpatient
20
15
10
5
0




```
print("Number of outliers (using IQR method):")

for col in numerical_cols_for_plotting:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
    num_outliers = outliers.shape[0]

    print(f"Column '{col}': {num_outliers} outliers")
```

```
Number of outliers (using IQR method):
Column 'time_in_hospital': 2252 outliers
Column 'num_lab_procedures': 143 outliers
Column 'num_procedures': 4954 outliers
Column 'num_medications': 2557 outliers
Column 'number_outpatient': 16739 outliers
```

```
Column 'number_emergency': 11383 outliers
Column 'number_inpatient': 7049 outliers
Column 'number_diagnoses': 281 outliers
```

We can see there are many outliers, but we won't replace all of them. For number_inpatient, number_emergency, number_outpatient the distribution is very tight around one number, if we would replace the outliers in these features, all the values would be the same. We would lose potential information. Therefore we will deal with outliers in number_diagnoses, num_lab_procedures, num_medications, num_procedures and time_in_hospital only, we will find outliers using IQR and then we will set them to the value of quantile bound.

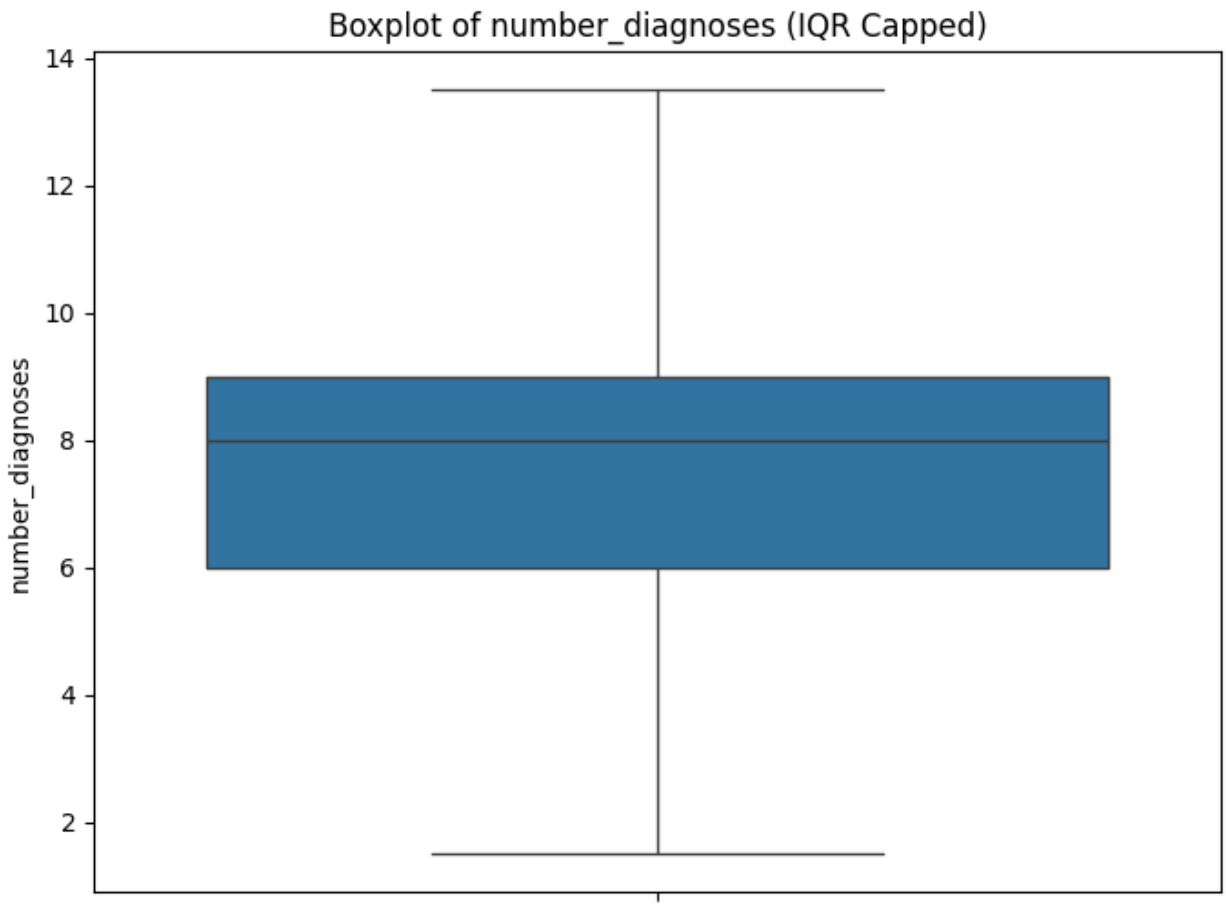
```
columns_to_cap_iqr = ['number_diagnoses', 'num_lab_procedures',
'num_medications', 'num_procedures', 'time_in_hospital']

for col in columns_to_cap_iqr:

    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df[col] = df[col].apply(lambda x: lower_bound if x < lower_bound
else x)

    df[col] = df[col].apply(lambda x: upper_bound if x > upper_bound
else x)

print("\nDescriptive Statistics for capped columns after IQR
capping:")
display(df[columns_to_cap_iqr].describe())
```

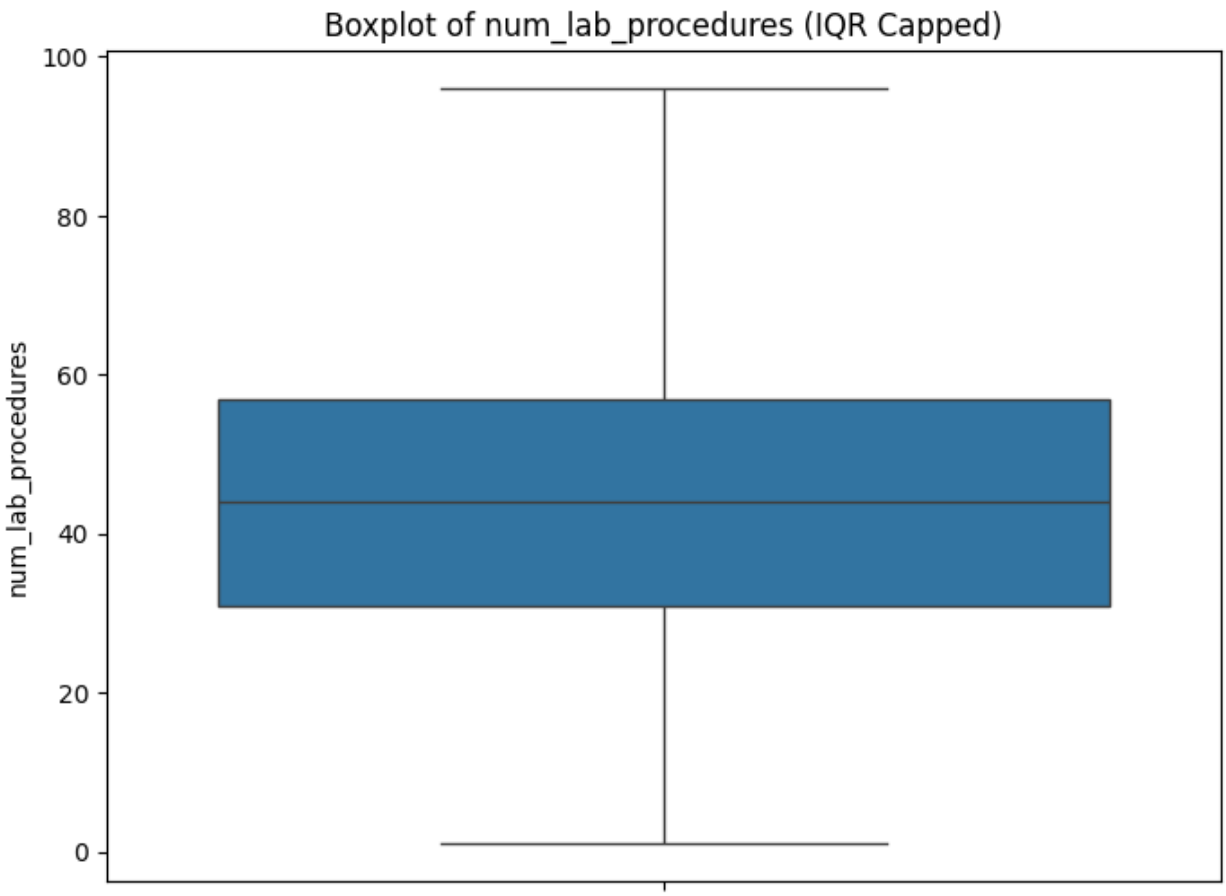
```
Descriptive Statistics for capped columns after IQR capping:

       number_diagnoses  num_lab_procedures  num_medications
num_procedures  \
count     101766.000000       101766.000000    101766.000000
101766.000000
mean           7.422395           43.087210        15.808512
1.291050
std            1.925325           19.648915         7.396645
1.581884
min            1.500000            1.000000         1.000000
0.000000
25%            6.000000           31.000000        10.000000
0.000000
50%            8.000000           44.000000        15.000000
1.000000
```

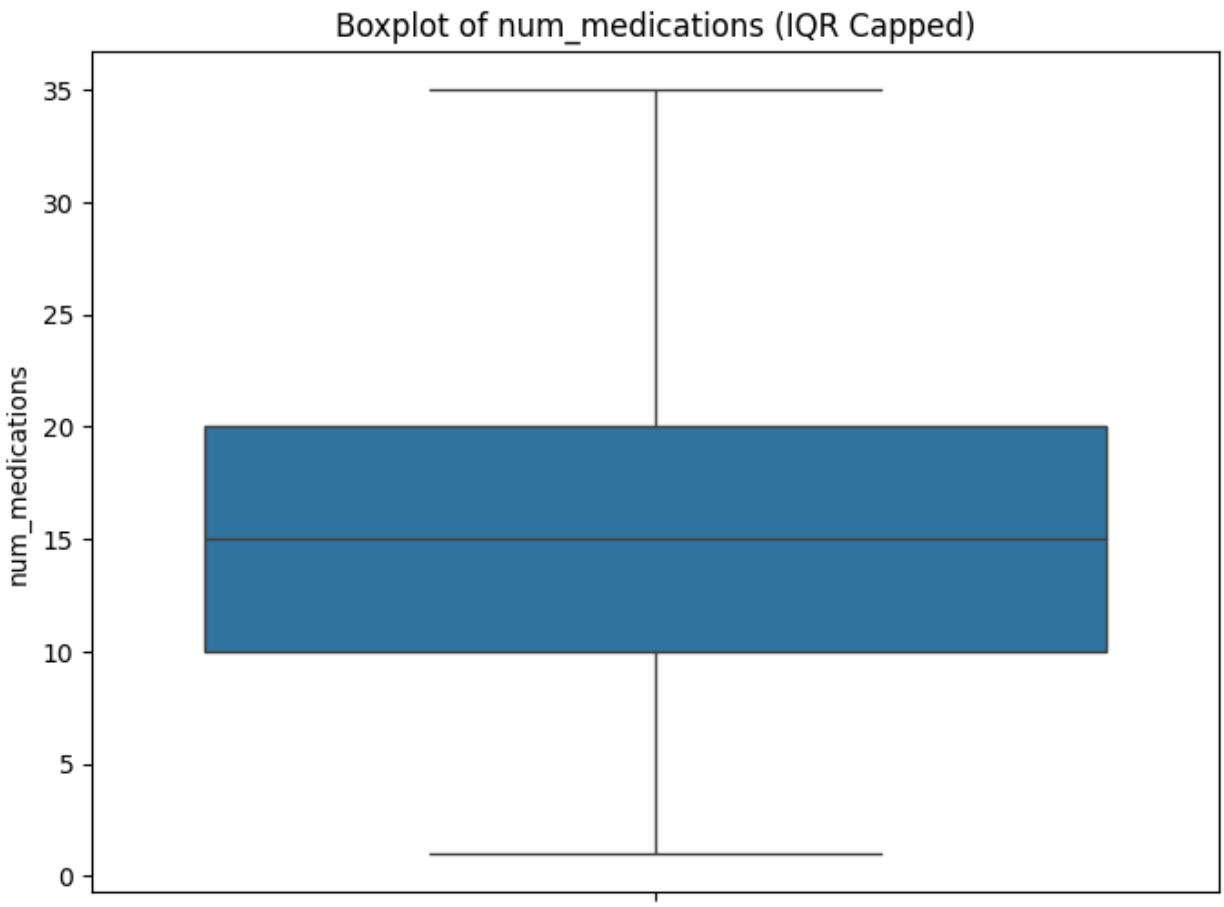
```
75%             9.000000            57.000000        20.000000
2.000000
max            13.500000            96.000000        35.000000
5.000000

       time_in_hospital
count     101766.000000
mean           4.363618
std            2.892181
min            1.000000
25%            2.000000
50%            4.000000
75%            6.000000
max           12.000000
```

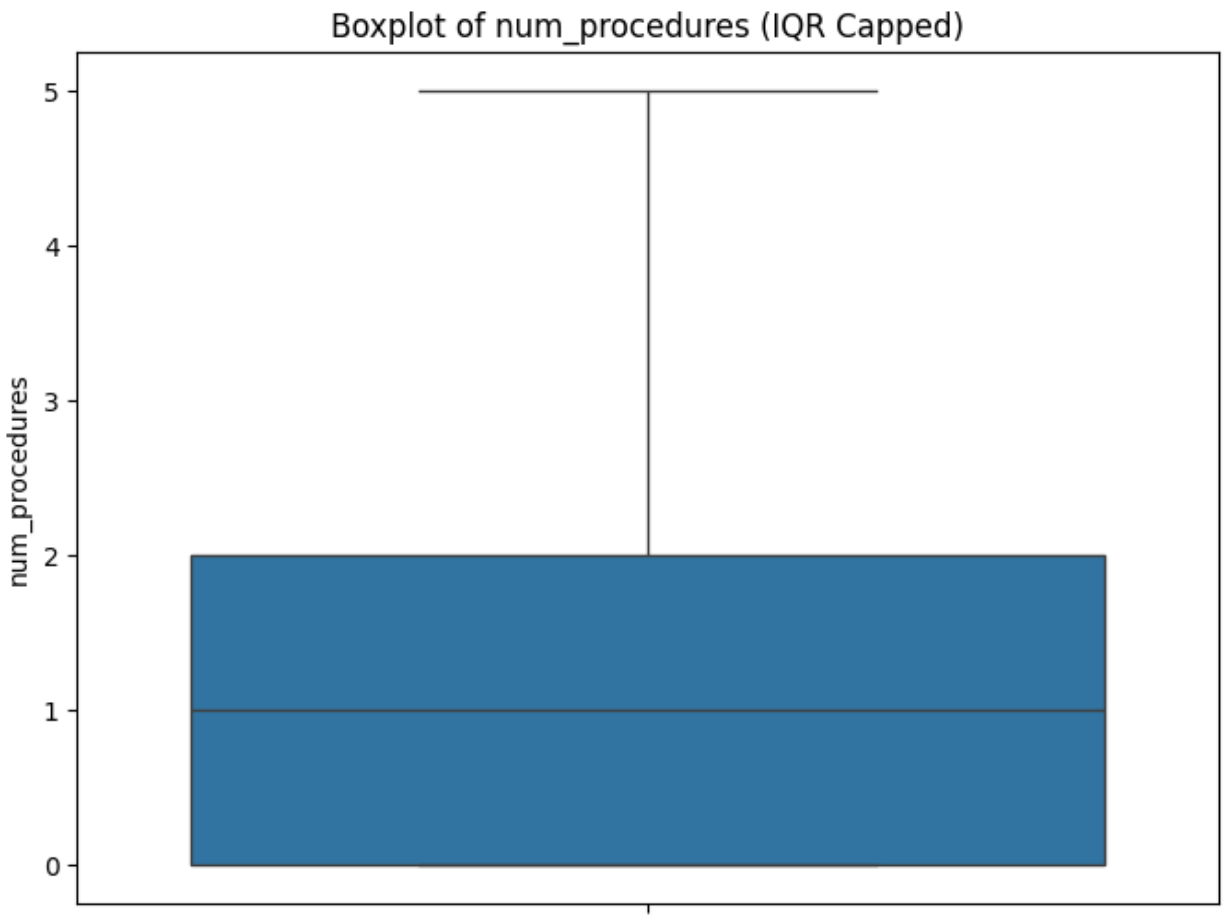
```
columns_to_plot_iqr_capped = ['number_diagnoses',
'num_lab_procedures', 'num_medications', 'num_procedures',
'time_in_hospital']

print("Boxplots for columns after IQR-based outlier capping:")

for col in columns_to_plot_iqr_capped:

    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df, y=col)
    plt.title(f'Boxplot of {col} (IQR Capped)')
    plt.ylabel(col)
    plt.show()
```

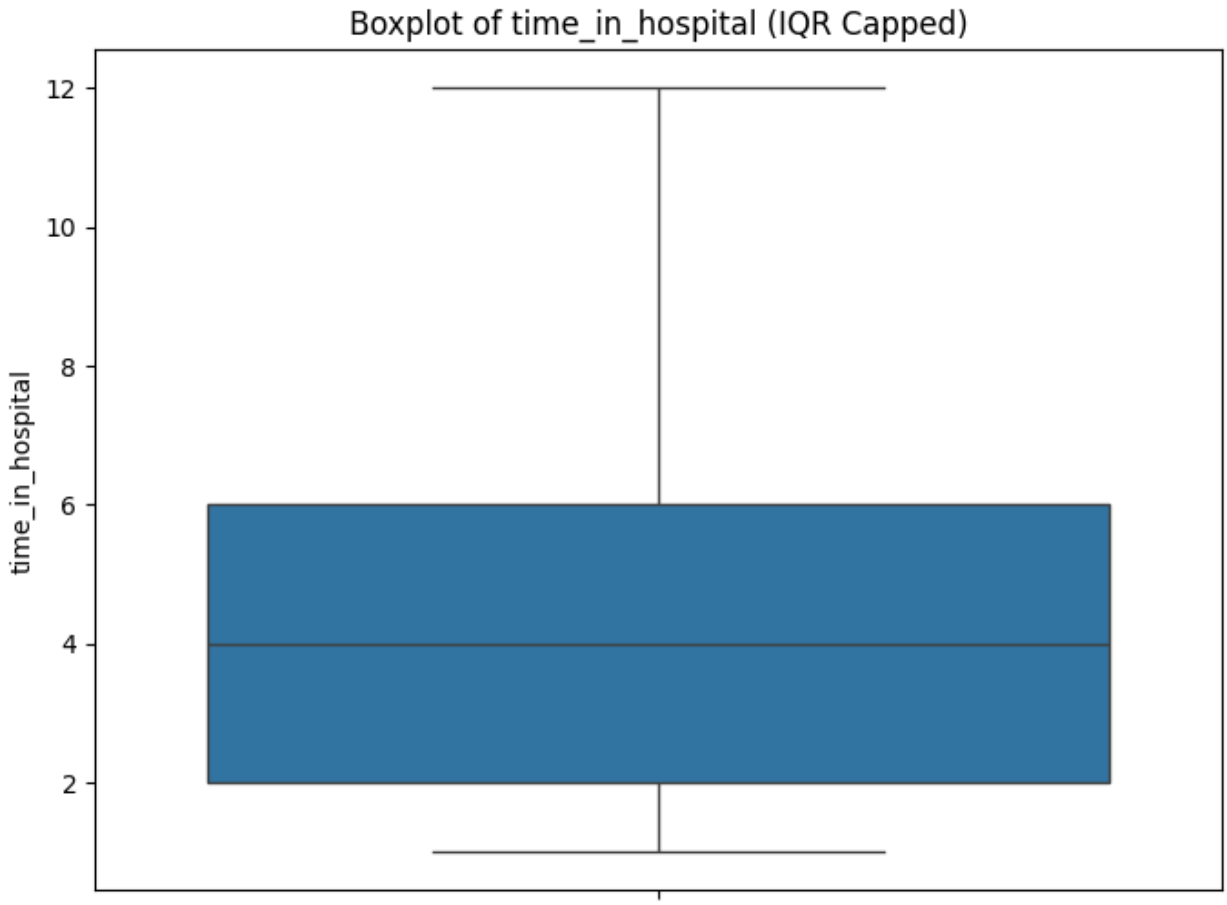
```
Boxplots for columns after IQR-based outlier capping:
```


Boxplot of number_diagnoses (IQR Capped)
number_diagnoses
14
12
10
8
6
4
2


Boxplot of num_lab_procedures (IQR Capped)
num_lab_procedures
100
80
60
40
20
0


Boxplot of num_medications (IQR Capped)
num_medications
35
30
25
20
15
10
5
0


Boxplot of num_procedures (IQR Capped)
num_procedures
0
1
2
3
4
5



Now we can see there are no outliers in these selected features.

At last we will scale the features to be in range of -1,1.

```
from sklearn.preprocessing import MinMaxScaler

numerical_cols = ['time_in_hospital', 'num_lab_procedures',
'num_procedures', 'num_medications', 'number_outpatient',
'number_emergency', 'number_inpatient', 'number_diagnoses']

scaler = MinMaxScaler()

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

print("Scaled numerical features:")
display(df[numerical_cols].head())
```

```
Scaled numerical features:

   time_in_hospital  num_lab_procedures  num_procedures
num_medications  \
0          0.272727            0.421053             0.4
0.470588
```

```
1          0.181818             0.663158              0.0
0.264706
2          0.090909             0.400000              0.0
0.205882
3          0.000000             0.400000              0.4
0.323529
4          0.181818             0.210526              0.4
0.647059

   number_outpatient  number_emergency  number_inpatient
number_diagnoses
0            0.00000               0.0          0.000000
0.625000
1            0.00000               0.0          0.047619
0.458333
2            0.00000               0.0          0.000000
0.041667
3            0.00000               0.0          0.047619
0.125000
4            0.02381               0.0          0.000000
0.625000
```

Next step is to encode categories to be represented by numbers only.

```
categorical_cols_for_encoding =
df.select_dtypes(include=['category']).columns.tolist()

cols_to_keep_all = [col for col in categorical_cols_for_encoding if
df[col].nunique() > 2]

cols_to_drop_first = [col for col in categorical_cols_for_encoding if
df[col].nunique() == 2]

df = pd.get_dummies(df, columns=cols_to_drop_first, drop_first=True,
prefix_sep='|')

df = pd.get_dummies(df, columns=cols_to_keep_all, drop_first=False,
prefix_sep='|')



print("Shape of new DataFrame:", df.shape)
print("\nColumns after one-hot encoding:")
print(df.columns)

df.head()
```

```
Shape of new DataFrame: (101766, 1159)

Columns after one-hot encoding:
```

```
Index(['time_in_hospital', 'num_lab_procedures', 'num_procedures',
       'num_medications', 'number_outpatient', 'number_emergency',
       'number_inpatient', 'number_diagnoses', 'diag|820', 'diag|276',
       ...
       'insulin|No', 'insulin|Steady', 'insulin|Up',
       'glyburide.metformin|Down', 'glyburide.metformin|No',
       'glyburide.metformin|Steady', 'glyburide.metformin|Up', 'class|
<30',
       'class|>30', 'class|NO'],
      dtype='object', length=1159)

   time_in_hospital  num_lab_procedures  num_procedures
num_medications  \
0          0.272727            0.421053             0.4
0.470588
1          0.181818            0.663158             0.0
0.264706
2          0.090909            0.400000             0.0
0.205882
3          0.000000            0.400000             0.4
0.323529
4          0.181818            0.210526             0.4
0.647059

   number_outpatient  number_emergency  number_inpatient
number_diagnoses  \
0            0.00000               0.0          0.000000
0.625000
1            0.00000               0.0          0.047619
0.458333
2            0.00000               0.0          0.000000
0.041667
3            0.00000               0.0          0.047619
0.125000
4            0.02381               0.0          0.000000
0.625000

   diag|820  diag|276  ...  insulin|No  insulin|Steady  insulin|Up  \
0         1         0  ...       False            True       False
1         0         1  ...       False            True       False
2         0         0  ...       False            True       False
3         0         0  ...        True           False       False
4         0         0  ...        True           False       False

   glyburide.metformin|Down  glyburide.metformin|No  \
0                     False                    True
1                     False                    True
2                     False                    True
3                     False                    True
4                     False                    True
```

```
   glyburide.metformin|Steady  glyburide.metformin|Up  class|<30
class|>30  \
0                         False                    False      False
False
1                         False                    False      False
False
2                         False                    False      False
False
3                         False                    False      False
False
4                         False                    False      False
False

   class|NO
0      True
1      True
2      True
3      True
4      True

[5 rows x 1159 columns]
```

```
encoded_columns = df.columns

feature_counts = {}

for col in encoded_columns:
    parts = col.split('|')

    if len(parts) > 1:
        original_feature = parts[0]

        if original_feature in feature_counts:
            feature_counts[original_feature] += 1
        else:
            feature_counts[original_feature] = 1


print("Number of encoded columns per original feature:")
for feature, count in feature_counts.items():
    print(f"{feature}: {count}")
```

```
Number of encoded columns per original feature:
diag: 915
acetohexamide: 1
tolbutamide: 1
troglitazone: 1
glipizide.metformin: 1
glimepiride.pioglitazone: 1
metformin.rosiglitazone: 1
```

```
metformin.pioglitazone: 1
change: 1
diabetesMed: 1
admission_type_id: 8
discharge_disposition_id: 26
admission_source_id: 17
race: 6
gender: 3
age: 10
payer_code: 18
medical_specialty: 73
max_glu_serum: 4
A1Cresult: 4
metformin: 4
repaglinide: 4
nateglinide: 4
chlorpropamide: 4
glimepiride: 4
glipizide: 4
glyburide: 4
pioglitazone: 4
rosiglitazone: 4
acarbose: 4
miglitol: 4
tolazamide: 3
insulin: 4
glyburide.metformin: 4
class: 3
```

Now we have our dataset with numeric features only, but it has 1159 features. We will try and drop features that have either small correlation or have low importance.

```
correlation_matrix = df.corrwith(df['diabetesMed|Yes'])

encoded_cols = [col for col in df.columns if df[col].dtype != 'float']
df[encoded_cols] = df[encoded_cols].astype(int)


print("\nCorrelation with 'diabetesMed|Yes':")
display(correlation_matrix.sort_values(ascending=False).head(30))
display(correlation_matrix.sort_values(ascending=True).head(30))
```

```
Correlation with 'diabetesMed|Yes':

diabetesMed|Yes                1.000000
insulin|Steady                 0.360434
metformin|Steady               0.256281
insulin|Down                   0.201861
glipizide|Steady               0.193680
```

```
num_medications                          0.193321
insulin|Up                               0.193296
glyburide|Steady                         0.173046
pioglitazone|Steady                      0.148253
rosiglitazone|Steady                     0.137996
glimepiride|Steady                       0.119850
A1Cresult|>8                             0.093123
repaglinide|Steady                       0.064168
time_in_hospital                         0.063169
diag|250.02                              0.058306
metformin|Up                             0.056254
glyburide|Up                             0.049011
glipizide|Up                             0.047717
class|>30                                0.046396
glyburide.metformin|Steady               0.045218
payer_code|MC                            0.044897
nateglinide|Steady                       0.044422
metformin|Down                           0.041195
glyburide|Down                           0.040797
glipizide|Down                           0.040651
medical_specialty|Emergency/Trauma       0.038697
payer_code|SP                            0.034813
payer_code|MD                            0.033967
num_lab_procedures                       0.033181
diag|250.13                              0.031863
dtype: float64
```

```
insulin|No                      -0.585464
change|No                       -0.506370
metformin|No                    -0.270176
glipizide|No                    -0.206230
glyburide|No                    -0.186835
pioglitazone|No                 -0.152230
rosiglitazone|No                -0.141157
glimepiride|No                  -0.126699
payer_code|Missing              -0.075663
discharge_disposition_id|18     -0.069842
A1Cresult|None                  -0.069683
repaglinide|No                  -0.067718
class|NO                        -0.061508
max_glu_serum|Norm              -0.046610
glyburide.metformin|No          -0.045677
nateglinide|No                  -0.045579
diag|560                        -0.041975
discharge_disposition_id|11     -0.033247
diag|562                        -0.032704
diag|789                        -0.031176
admission_type_id|6             -0.030737
payer_code|HM                   -0.030696
acarbose|No                     -0.030110
```

```
age|[90-100)                    -0.029543
admission_source_id|5          -0.027727
diag|578                       -0.027317
diag|553                       -0.024053
diag|574                       -0.023991
medical_specialty|Missing      -0.023135
diag|558                       -0.022460
dtype: float64
```

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X = df.drop(columns=['diabetesMed|Yes'])
y = df['diabetesMed|Yes']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

model = RandomForestClassifier(n_estimators=200, random_state=42,
n_jobs=-1)
model.fit(X_train, y_train)

importances = pd.Series(model.feature_importances_, index=X.columns)

importances_sorted = importances.sort_values(ascending=False)
print(importances_sorted.head(50))
```

```
insulin|No                     0.213665
insulin|Steady                 0.113936
change|No                      0.099622
metformin|No                   0.054329
metformin|Steady               0.046696
glipizide|No                   0.037536
glipizide|Steady               0.036931
glyburide|No                   0.035107
glyburide|Steady               0.031467
insulin|Down                   0.024147
insulin|Up                     0.021271
pioglitazone|No                0.017476
pioglitazone|Steady            0.017219
num_medications                0.016498
glimepiride|Steady             0.014636
glimepiride|No                 0.014391
rosiglitazone|No               0.013940
rosiglitazone|Steady           0.013675
num_lab_procedures             0.008100
time_in_hospital               0.005927
number_diagnoses               0.004706
repaglinide|No                 0.004435
```

```
num_procedures                  0.003829
repaglinide|Steady              0.003725
number_inpatient                0.003229
A1Cresult|>8                    0.002546
payer_code|Missing              0.002486
glyburide.metformin|No          0.002328
number_outpatient               0.002222
glyburide.metformin|Steady      0.002204
class|NO                        0.002180
nateglinide|No                  0.001763
medical_specialty|Missing       0.001742
nateglinide|Steady              0.001690
diag|250                        0.001687
number_emergency                0.001676
age|[70-80)                     0.001670
gender|Male                     0.001650
A1Cresult|None                  0.001636
gender|Female                   0.001619
class|>30                       0.001573
discharge_disposition_id|1      0.001569
race|Caucasian                  0.001503
payer_code|MC                   0.001457
discharge_disposition_id|18     0.001454
age|[80-90)                     0.001407
age|[60-70)                     0.001377
glyburide|Up                    0.001376
admission_type_id|1             0.001370
diag|428                        0.001364
dtype: float64

correlated_features = correlation_matrix[abs(correlation_matrix) >
0.05].index.tolist()

important_features = importances_sorted[importances_sorted >
0.005].index.tolist()

selected_features = list(set(correlated_features) |
set(important_features))

if 'diabetesMed|Yes' in selected_features:
    selected_features.remove('diabetesMed|Yes')

print("Features with absolute correlation > 0.05:")
print(correlated_features)
print(f"\nNumber of selected features: {len(correlated_features)}")
print("\nFeatures with importance > 0.005:")
print(important_features)
print(f"\nNumber of selected features: {len(important_features)}")
print("\nSelected features (meeting both criteria):")
```

```
print(selected_features)
print(f"\nNumber of selected features: {len(selected_features)}")
```

```
Features with absolute correlation > 0.05:
['time_in_hospital', 'num_medications', 'diag|250.02', 'change|No',
'diabetesMed|Yes', 'discharge_disposition_id|18', 'payer_code|
Missing', 'A1Cresult|>8', 'A1Cresult|None', 'metformin|No',
'metformin|Steady', 'metformin|Up', 'repaglinide|No', 'repaglinide|
Steady', 'glimepiride|No', 'glimepiride|Steady', 'glipizide|No',
'glipizide|Steady', 'glyburide|No', 'glyburide|Steady', 'pioglitazone|
No', 'pioglitazone|Steady', 'rosiglitazone|No', 'rosiglitazone|
Steady', 'insulin|Down', 'insulin|No', 'insulin|Steady', 'insulin|Up',
'class|NO']

Number of selected features: 29

Features with importance > 0.005:
['insulin|No', 'insulin|Steady', 'change|No', 'metformin|No',
'metformin|Steady', 'glipizide|No', 'glipizide|Steady', 'glyburide|
No', 'glyburide|Steady', 'insulin|Down', 'insulin|Up', 'pioglitazone|
No', 'pioglitazone|Steady', 'num_medications', 'glimepiride|Steady',
'glimepiride|No', 'rosiglitazone|No', 'rosiglitazone|Steady',
'num_lab_procedures', 'time_in_hospital']

Number of selected features: 20

Selected features (meeting both criteria):
['metformin|Steady', 'repaglinide|Steady', 'time_in_hospital',
'num_medications', 'A1Cresult|None', 'A1Cresult|>8', 'diag|250.02',
'glipizide|Steady', 'glimepiride|No', 'metformin|Up', 'rosiglitazone|
No', 'insulin|Steady', 'glimepiride|Steady', 'num_lab_procedures',
'payer_code|Missing', 'rosiglitazone|Steady', 'pioglitazone|No',
'glyburide|No', 'pioglitazone|Steady', 'insulin|Up', 'metformin|No',
'discharge_disposition_id|18', 'class|NO', 'glipizide|No', 'insulin|
No', 'insulin|Down', 'glyburide|Steady', 'repaglinide|No', 'change|
No']

Number of selected features: 29
```

We have chosen to select features that have either correlation higher than 0.05 or importance higher then 0.005. This gives us 29 features

```
X = df[selected_features]
y = df['diabetesMed|Yes']
dataset = pd.concat([X, y], axis=1)
dataset.head()
```

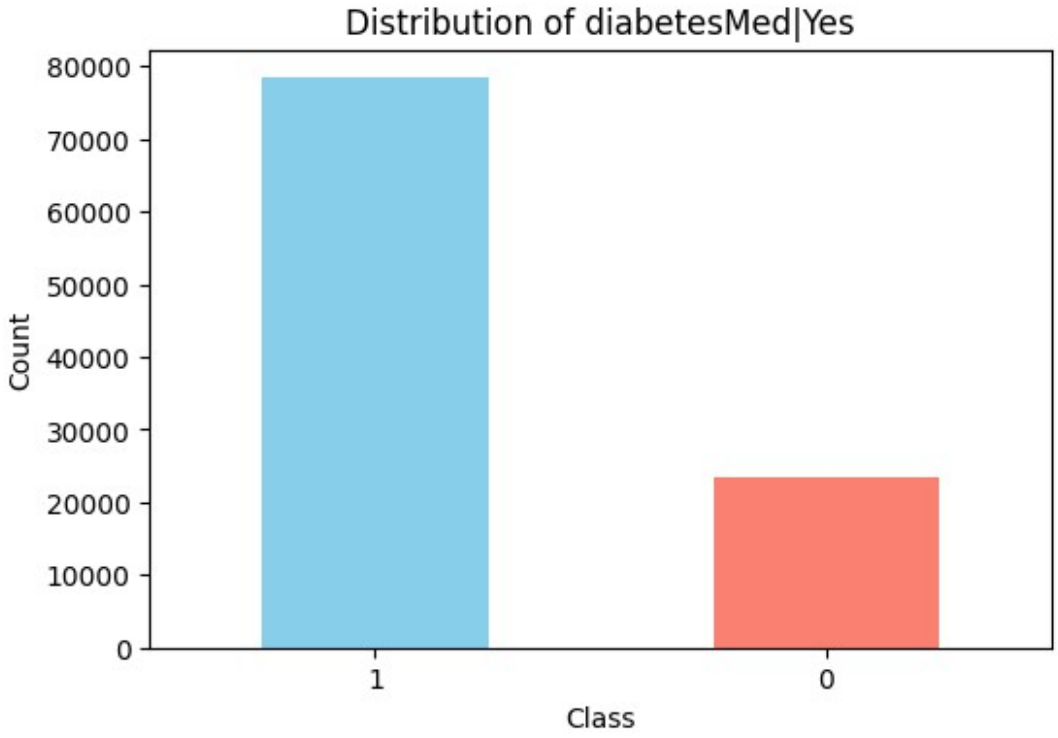
```
   metformin|Steady  repaglinide|Steady  time_in_hospital
num_medications  \
0                 0                   0          0.272727
```

```
0.470588
1                  0                    0          0.181818
0.264706
2                  1                    0          0.090909
0.205882
3                  0                    0          0.000000
0.323529
4                  1                    0          0.181818
0.647059

   A1Cresult|None  A1Cresult|>8  diag|250.02  glipizide|Steady  \
0               1              0            0                  1
1               1              0            0                  0
2               1              0            0                  0
3               1              0            0                  0
4               1              0            0                  0

   glimepiride|No  metformin|Up  ...  metformin|No  \
0               1              0  ...             1
1               1              0  ...             1
2               1              0  ...             0
3               1              0  ...             1
4               1              0  ...             0

   discharge_disposition_id|18  class|NO  glipizide|No  insulin|No  \
0                            0         1             0           0
1                            0         1             1           0
2                            0         1             1           0
3                            0         1             1           1
4                            0         1             1           1

   insulin|Down  glyburide|Steady  repaglinide|No  change|No
diabetesMed|Yes
0             0                  0               1          0
1
1             0                  1               1          0
1
2             0                  0               1          0
1
3             0                  0               1          1
0
4             0                  1               1          0
1

[5 rows x 30 columns]
```

```
class_counts = dataset['diabetesMed|Yes'].value_counts()
print("Class distribution:\n", class_counts)

# --- Plot distribution ---
```

```
plt.figure(figsize=(6,4))
class_counts.plot(kind='bar', color=['skyblue','salmon'])
plt.title('Distribution of diabetesMed|Yes')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```

```
Class distribution:
 diabetesMed|Yes
1    78363
0    23403
Name: count, dtype: int64
```



Dataset is imbalanced, therefore we will punish misclassification of the minority group more heavily.

```
dataset.to_csv('../processed dataset/processed_dataset.csv',
index=False)
```

Experiments

After testing the model with this dataframe, the results were too accurate, meaning there were likely features that correlated too highly with the predicted feature, meaning the model could (and did) use these features to quickly and accurately get the correct prediction.

After further research we found out that these features were all types of diabetes medication: insulin, metformin, glyburide, glipizide, glyburide.metformin, nateglinide, pioglitazone, rosiglitazone, glimepiride, repaglinide, acarbose, miglitol, troglitazone, tolazamide, tolbutamide, chlorpropamide, sitagliptin.

Perform another data selection with different features. (The same as in dataloader.py)

```
import pandas as pd
import numpy as np
import torch
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import openml

dataset_id = 45069
test_size = 0.2
random_state = 42
```

Load the data.

```
dataset = openml.datasets.get_dataset(dataset_id)
X, y, _, _ = dataset.get_data(
    dataset_format="dataframe",
    target=dataset.default_target_attribute
)

if y is not None:
    df = pd.concat([X, y], axis=1)
else:
    df = X

print(f"Loaded {df.shape[0]} records with {df.shape[1]} features")
```

```
Loaded 101766 records with 48 features
```

Drop unnecessary columns.

```
columns_to_drop = ['examide', 'citoglipton', 'weight']

df = df.drop(columns=columns_to_drop)
print(f"Dropped columns: {columns_to_drop}")
```

```
Dropped columns: ['examide', 'citoglipton', 'weight']
```

Convert to categorical columns.

```
columns_to_convert = [
    'admission_type_id',
    'discharge_disposition_id',
    'admission_source_id',
    'class',
    'diabetesMed',
]
for col in columns_to_convert:
    if col in df.columns:
        df[col] = df[col].astype('category')
```

Handle missing (NaN) values by replacing them with a new category called 'Missing'.

```
categorical_cols_with_missing = ['payer_code', 'medical_specialty']

for col in categorical_cols_with_missing:
    if col in df.columns:
        if pd.api.types.is_categorical_dtype(df[col].dtype):
            if 'Missing' not in df[col].cat.categories:
                df[col] = df[col].cat.add_categories('Missing')
        df[col] = df[col].fillna('Missing')
```

```
C:\Users\maxik\AppData\Local\Temp\ipykernel_38132\1295604742.py:5:
DeprecationWarning: is_categorical_dtype is deprecated and will be
removed in a future version. Use isinstance(dtype,
pd.CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(df[col].dtype):
```

Create unified diagnosis columns using all 3 current diagnosis columns.

```
all_diag_codes = pd.concat([
    df['diag_1'],
    df['diag_2'],
    df['diag_3']
]).unique()
all_diag_codes = [code for code in all_diag_codes if pd.notna(code)]

print(f"Creating {len(all_diag_codes)} diagnosis binary columns...")

diag_cols_dict = {
    f'diag|{code}': (
        (df['diag_1'] == code) |
        (df['diag_2'] == code) |
        (df['diag_3'] == code)
    ).astype(int)
    for code in all_diag_codes
}
```

```
diag_cols_df = pd.DataFrame(diag_cols_dict, index=df.index)
df = df.drop(columns=['diag_1', 'diag_2', 'diag_3'])
df = pd.concat([df, diag_cols_df], axis=1)
```

```
Creating 915 diagnosis binary columns...
```

Cap outliers using the IQR method.

```
columns_to_cap = [
    'number_diagnoses',
    'num_lab_procedures',
    'num_medications',
    'num_procedures',
    'time_in_hospital'
]

for col in columns_to_cap:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    df[col] = df[col].apply(
        lambda x: lower_bound if x < lower_bound else (
            upper_bound if x > upper_bound else x
        )
    )

print(f"Capped outliers for {len(columns_to_cap)} columns")
```

```
Capped outliers for 5 columns
```

One-hot encode the columns.

```
categorical_cols =
df.select_dtypes(include=['category']).columns.tolist()

cols_to_drop_first = [
    col for col in categorical_cols
    if df[col].nunique() == 2
]

cols_to_keep_all = [
    col for col in categorical_cols
    if df[col].nunique() > 2
]
```

```
df = pd.get_dummies(
    df,
    columns=cols_to_drop_first,
    drop_first=True,
    prefix_sep='|'
)
df = pd.get_dummies(
    df,
    columns=cols_to_keep_all,
    drop_first=False,
    prefix_sep='|'
)

encoded_cols = [
    col for col in df.columns
    if df[col].dtype != 'float'
]
df[encoded_cols] = df[encoded_cols].astype(int)

print(f"One-hot encoded: {df.shape[1]} columns")
```

```
One-hot encoded: 1158 columns
```

Select features based on a correlation_threshold = 0.05 and an importance_threshold = 0.005

```
correlation_threshold = 0.05
importance_threshold = 0.005

y = df['diabetesMed|Yes']
diabetes_cols = [c for c in df.columns if
c.startswith('diabetesMed|')]
X = df.drop(columns=diabetes_cols)

correlation_matrix = df.corrwith(df['diabetesMed|Yes'])
correlated_features = correlation_matrix[
    abs(correlation_matrix) > correlation_threshold
].index.tolist()

X_train_fs, X_test_fs, y_train_fs, y_test_fs = train_test_split(
    X, y, test_size=test_size, random_state=random_state
)

rf_model = RandomForestClassifier(
    n_estimators=200,
    random_state=random_state,
    n_jobs=-1
)
rf_model.fit(X_train_fs, y_train_fs)

importances = pd.Series(rf_model.feature_importances_,
```

```
index=X.columns)
important_features = importances[
    importances > importance_threshold
].index.tolist()

selected_features = list(set(correlated_features) |
set(important_features))
selected_features = [f for f in selected_features if not
f.startswith('diabetesMed|')]

print(f"Selected {len(selected_features)} features")
print(f"  - Correlated features: {len(correlated_features)}")
print(f"  - Important features: {len(important_features)}")
```

```
Selected 29 features
  - Correlated features: 29
  - Important features: 20
```

Prepare data and drop duplicates.

```
X = df[selected_features]
y = df['diabetesMed|Yes']

print(f"Shape before dropping duplicates: {X.shape}")
X = X.drop_duplicates()
y = y.loc[X.index]
print(f"Shape after dropping duplicates: {X.shape}")
```

```
Shape before dropping duplicates: (101766, 29)
Shape after dropping duplicates: (92766, 29)
```

Create train-test split.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=test_size,
    random_state=random_state,
    stratify=y
)

X_train = torch.tensor(X_train.to_numpy(), dtype=torch.float32)
y_train = torch.tensor(y_train.to_numpy(),
dtype=torch.float32).unsqueeze(1)
X_test = torch.tensor(X_test.to_numpy(), dtype=torch.float32)
y_test = torch.tensor(y_test.to_numpy(),
dtype=torch.float32).unsqueeze(1)

print(f"\nFinal tensor shapes:")
print(f"  X_train: {X_train.shape}")
```

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


Final tensor shapes:
  X_train: torch.Size([74212, 29])
  X_test: torch.Size([18554, 29])
  y_train: torch.Size([74212, 1])
  y_test: torch.Size([18554, 1])
```

Select features based on correlation and RandomForest importance using only training data.

```
def select_features_train(X_train_df: pd.DataFrame, y_train_s:
pd.Series,
                                correlation_threshold: float = 0.05,
                                importance_threshold: float = 0.005) -
> list:

        std = X_train_df.std(numeric_only=True)
        non_constant_cols = std[std > 0].index
        dropped_zero_var = X_train_df.shape[1] -
len(non_constant_cols)
        if dropped_zero_var > 0:
            print(f"Dropped {dropped_zero_var} zero-variance columns
from train before correlation")
        X_train_nc = X_train_df[non_constant_cols]

        with np.errstate(invalid='ignore', divide='ignore'):
            corr =
X_train_nc.corrwith(y_train_s).abs().fillna(0.0).sort_values(ascending
=False)
        corr_features = corr[corr >
correlation_threshold].index.tolist()

        rf_model = RandomForestClassifier(n_estimators=200,
random_state=random_state, n_jobs=-1)
        rf_model.fit(X_train_nc, y_train_s)
        importances = pd.Series(rf_model.feature_importances_,
index=X_train_nc.columns)
        imp_features = importances[importances >
importance_threshold].index.tolist()

        selected = list(set(corr_features) | set(imp_features))
        print(f"Selected {len(selected)} features (train-only)")
        print(f"  - Correlated: {len(corr_features)}  Important:
{len(imp_features)}")
        return selected
```

Load and preprocess the data.

```
# Define target and remove leaky columns
if 'diabetesMed|Yes' not in df.columns:
    raise ValueError("Target column 'diabetesMed|Yes' not found after
encoding.")

y_full = df['diabetesMed|Yes']

# Drop any one-hot columns derived from specific diabetes medications
or the target/change
leaky_prefixes = [
    'diabetesMed', 'change', 'insulin', 'metformin', 'glyburide',
'glipizide',
    'glyburide.metformin', 'nateglinide', 'pioglitazone',
'rosiglitazone',
    'glimepiride', 'repaglinide', 'acarbose', 'miglitol',
'troglitazone',
    'tolazamide', 'tolbutamide', 'chlorpropamide', 'sitagliptin'
]
leaky_cols = [c for c in df.columns if any(c.startswith(prefix + '|')
for prefix in leaky_prefixes)]
if leaky_cols:
    print(f"Dropping medication-related columns: {len(leaky_cols)}")
X_full = df.drop(columns=leaky_cols + ['diabetesMed|Yes'])

# Train/Test split
X_train_df, X_test_df, y_train_s, y_test_s = train_test_split(
    X_full, y_full, test_size=test_size, random_state=random_state,
stratify=y_full
)

# Feature selection using only training data
selected = select_features_train(X_train_df, y_train_s)
selected_features = selected

# Apply selected features to both splits
X_train_df = X_train_df[selected]
X_test_df = X_test_df[selected]

# Optionally drop duplicates on train only
print(f"Shape before dropping duplicates (train): {X_train_df.shape}")
X_train_df = X_train_df.drop_duplicates()
y_train_s = y_train_s.loc[X_train_df.index]
print(f"Shape after  dropping duplicates (train): {X_train_df.shape}")

# Convert to tensors
X_train = torch.tensor(X_train_df.to_numpy(), dtype=torch.float32)
y_train = torch.tensor(y_train_s.to_numpy(),
dtype=torch.float32).unsqueeze(1)
X_test = torch.tensor(X_test_df.to_numpy(), dtype=torch.float32)
y_test = torch.tensor(y_test_s.to_numpy(),
```

```
dtype=torch.float32).unsqueeze(1)

n_features = X_train.shape[1]

print("="*80)
print(f"PREPROCESSING COMPLETE - {n_features} features selected
(train-only selection, no leakage)")
print("="*80)
```

```
Dropping leaky medication-related columns: 59
Dropped 19 zero-variance columns from train before correlation
Selected 47 features (train-only)
  - Correlated: 8  Important: 45
Shape before dropping duplicates (train): (81412, 47)
Shape after  dropping duplicates (train): (81406, 47)
================================================================================
==========
PREPROCESSING COMPLETE - 47 features selected (train-only selection,
no leakage)
================================================================================
==========
```

Final dataframe inspection.

```
print(f"\n{'='*80}")
print("DATA SHAPES")
print(f"{'='*80}")
print(f"X_train: {X_train.shape}")
print(f"X_test:  {X_test.shape}")
print(f"y_train: {y_train.shape}")
print(f"y_test:  {y_test.shape}")
print(f"Number of features (n_input): {n_features}")

# Get feature names
if selected_features is not None:
    feature_names = selected_features
    print(f"\n{'='*80}")
    print(f"SELECTED FEATURES ({len(feature_names)} total)")
    print(f"{'='*80}")
    for i, feat in enumerate(feature_names, 1):
        print(f"{i:2d}. {feat}")
else:
    print("\nWarning: Could not retrieve feature names")

# Show class distribution
print(f"\n{'='*80}")
print("TARGET DISTRIBUTION")
print(f"{'='*80}")
import torch
```

```
if isinstance(y_train, torch.Tensor):
    y_train_np = y_train.cpu().numpy()
    y_test_np = y_test.cpu().numpy()
else:
    y_train_np = y_train
    y_test_np = y_test

train_pos = int(y_train_np.sum())
train_total = len(y_train_np)
test_pos = int(y_test_np.sum())
test_total = len(y_test_np)

print(f"Train: {train_pos}/{train_total}
({100*train_pos/train_total:.1f}%) positive")
print(f"Test:  {test_pos}/{test_total} ({100*test_pos/test_total:.1f}
%) positive")

# Show basic stats for X_train
print(f"\n{'='*80}")
print("FEATURE STATISTICS (Training Set)")
print(f"{'='*80}")

if isinstance(X_train, torch.Tensor):
    X_train_np = X_train.cpu().numpy()
else:
    X_train_np = X_train

df_stats = pd.DataFrame({
    'Feature': feature_names if 'feature_names' in locals() else
[f"Feature_{i}" for i in range(n_features)],
    'Mean': X_train_np.mean(axis=0),
    'Std': X_train_np.std(axis=0),
    'Min': X_train_np.min(axis=0),
    'Max': X_train_np.max(axis=0),
    'Non-zero %': 100 * (X_train_np != 0).sum(axis=0) /
len(X_train_np),
})

# Show top 20 features
print("\nTop 20 features:")
print(df_stats.head(20).to_string(index=False))

print(f"\n{'='*80}")
print("INSPECTION COMPLETE")
print(f"{'='*80}")


================================================================================
==========
DATA SHAPES
```

```
================================================================================
==========
X_train: torch.Size([81406, 47])
X_test:  torch.Size([20354, 47])
y_train: torch.Size([81406, 1])
y_test:  torch.Size([20354, 1])
Number of features (n_input): 47

================================================================================
==========
SELECTED FEATURES (47 total)
================================================================================
==========
 1. time_in_hospital
 2. number_emergency
 3. age|[70-80)
 4. discharge_disposition_id|18
 5. diag|414
 6. discharge_disposition_id|3
 7. payer_code|Missing
 8. admission_type_id|3
 9. class|>30
10. diag|403
11. age|[40-50)
12. num_procedures
13. diag|276
14. age|[80-90)
15. diag|250.02
16. age|[60-70)
17. race|AfricanAmerican
18. diag|428
19. gender|Male
20. race|Caucasian
21. age|[50-60)
22. diag|427
23. diag|780
24. gender|Female
25. num_medications
26. admission_type_id|2
27. discharge_disposition_id|1
28. diag|786
29. diag|496
30. diag|250
31. A1Cresult|>8
32. discharge_disposition_id|6
33. number_inpatient
34. medical_specialty|Missing
35. medical_specialty|Family/GeneralPractice
36. medical_specialty|InternalMedicine
```

```
37. A1Cresult|None
38. number_outpatient
39. class|NO
40. number_diagnoses
41. diag|401
42. admission_source_id|7
43. admission_source_id|1
44. diag|599
45. payer_code|MC
46. num_lab_procedures
47. admission_type_id|1

================================================================================
TARGET DISTRIBUTION
================================================================================
Train: 62685/81406 (77.0%) positive
Test:  15673/20354 (77.0%) positive

================================================================================
FEATURE STATISTICS (Training Set)
================================================================================

Top 20 features:
                     Feature     Mean      Std  Min  Max  Non-zero %
           time_in_hospital 4.360784 2.887990  1.0 12.0  100.000000
           number_emergency 0.196226 0.924113  0.0 76.0   11.150284
                age|[70-80) 0.256492 0.436697  0.0  1.0   25.649215
discharge_disposition_id|18 0.036201 0.186791  0.0  1.0    3.620126
                   diag|414 0.120937 0.326054  0.0  1.0   12.093703
 discharge_disposition_id|3 0.137373 0.344241  0.0  1.0   13.737317
         payer_code|Missing 0.394909 0.488831  0.0  1.0   39.490947
        admission_type_id|3 0.185773 0.388923  0.0  1.0   18.577255
                  class|>30 0.348328 0.476441  0.0  1.0   34.832813
                   diag|403 0.055868 0.229667  0.0  1.0    5.586812
                age|[40-50) 0.095521 0.293933  0.0  1.0    9.552121
            num_procedures 1.285028 1.577680  0.0  5.0   54.090608
                   diag|276 0.127312 0.333323  0.0  1.0   12.731248
                age|[80-90) 0.168956 0.374713  0.0  1.0   16.895561
                diag|250.02 0.040513 0.197159  0.0  1.0    4.051298
                age|[60-70) 0.220303 0.414451  0.0  1.0   22.030317
       race|AfricanAmerican 0.188414 0.391042  0.0  1.0   18.841363
                   diag|428 0.171412 0.376869  0.0  1.0   17.141243
                gender|Male 0.463185 0.498643  0.0  1.0   46.318453
             race|Caucasian 0.747832 0.434257  0.0  1.0   74.783186

================================================================================
```

```
==========
INSPECTION COMPLETE
================================================================================
==========
```

The transformations showcased above are called before training the model. We load, transform data and select features using dataloader class and then we train the model with trainer and model classes.

Configuration

Model is configured using configuration file. These configuration files are contained inside .scratch/experiment_configs folder. For each experiment we create a file and then we run tools.main. This calls run command which logs the run to wandb.

Example of config file:

n_input: 47 layers: [512, 256, 128, 32, 8, 1]
activations: [ReLU, ReLU, ReLU, ReLU, ReLU, null]
output_activation: null
dropout: 0.1
batch_norm: true
skip_connections: false
bottleneck: true

training:
  epochs: 100
  lr: 0.0005
  batch_size: 256
  verbose: true
  patience: 10
  min_delta: 0.001

Explanation:

n_input - Number of input features fed into the first layer of the network.
layers - List defining the number of neurons in each hidden and output layer.
activations - Activation functions applied after each layer (e.g., ReLU, GELU).
output_activation - Final activation used for the output layer (e.g., Sigmoid for binary tasks).
dropout - Fraction of neurons randomly turned off during training to prevent overfitting.
batch_norm - Whether to apply Batch Normalization for stabilizing training.
skip_connections - Lets the model reuse information from earlier layers to learn better and faster.
bottleneck - Adds smaller intermediate layers to force the network to learn compact representations.

epochs - Maximum number of full passes through the training dataset.
lr - Learning rate controlling how much weights are updated per step.
batch_size - Number of samples processed together before an optimizer update.
verbose - Whether to print progress and loss values during training.

patience - Number of epochs to wait for improvement before early stopping.
min_delta - Minimum change in validation loss to be considered an improvement.

Experiment tracking

For our experiment tracking we use wandb. We call run from main and this saves the results of the tests.

Picture of runs

Confusion table

Learning process1

Learning process2

We can also compare the last 10 runs to see if we are improving.

Learning processes comparison

Experiment

For our experiments we have tried to change batch sizes, number of layers, applying optimization techniques - Dropout, Skip Connections, Bottleneck Layers. We have also tried changing the learning rate, patience and min_delta.

With prepared data we were able to train the model in 12 eepochs to:

test/accuracy:0.9956273951066128
test/f1:0.9971526378091308
test/precision:1
test/roc_auc:0.997850680470756
test/sensitivity:0.9943214445224272
test/specificity:1

Confusion Matrix

Confusion Matrix

We have used this configuration file:

n_input: 29 layers: [128, 64, 32, 1]
activations: [ReLU, ReLU, ReLU, null]
output_activation: null
dropout: 0.3
batch_norm: true
skip_connections: false
bottleneck: true

training:
  epochs: 100
  lr: 0.001
  batch_size: 256

verbose: true
patience: 10
min_delta: 0.001

Process of training:

Learning processes comparison

This result was expected, the high precision was achieved by including data about medications, which closely relates to if patient is using medications. If person is using any concrete medication for diabetes they are using medication for diabetes for sure.

To have a model with relevant purpuse we have removed these medications from our data. We have removed 'diabetesMed', 'change', 'insulin', 'metformin', 'glyburide', 'glipizide','glyburide.metformin', 'nateglinide', 'pioglitazone', 'rosiglitazone','glimepiride', 'repaglinide', 'acarbose', 'miglitol', 'troglitazone','tolazamide', 'tolbutamide', 'chlorpropamide', 'sitagliptin' which are medications that treat diabetes (change refers to change in medications). Next we did experiments using these new data.

For these we have found 3 models that can be thought of as the best models, first is best overall, second one is best when we want to minimize detection of diabetes with non-diabetic people and third when we want to prioritize detection of diabetes with diabetic people.

1st model

For this model were able to achieve in 100 epochs:

test/accuracy:0.6550555173430284
test/f1:0.7451265110538353
test/precision:0.8643254168772108
test/roc_auc:0.7076594152879849
test/sensitivity:0.654820391756524
test/specificity:0.6558427686391797
train/loss:0.27352656585709106

This model has accuracy of 65% and loss of 0.2735. We were not able to find better model.

Confusion matrix:

Confusion Matrix

We have used this configuration file:

n_input: 47 layers: [90, 29, 1]
activations: [ReLU, ReLU, null]
output_activation: null
dropout: 0.2
batch_norm: true
skip_connections: false
bottleneck: true

training:
epochs: 100

lr: 0.01
batch_size: 128
verbose: true
patience: 10
min_delta: 0.001

Process of training:

Learning processes comparison

2nd model

For this model were able to achieve in 54 epochs:

test/accuracy:0.47150437260489336
test/f1:0.5008121026497749
test/precision:0.9183117767188564
test/roc_auc:0.7211868025425039
test/sensitivity:0.3442863523256556
test/specificity:0.8974578081606495
train/loss:0.26762372878549984

This model has low accuracy but high precision.

Confusion matrix:

Confusion Matrix

We have used this configuration file:

n_input: 80 layers: [128,64,32,16,1]
activations: [ReLU, ReLU, ReLU, ReLU,null]
output_activation: null
dropout: 0.1
batch_norm: true
skip_connections: false
bottleneck: true

training:
epochs: 100
lr: 0.0001
batch_size: 128
verbose: true
patience: 30
min_delta: 0.001

Process of training:

Learning processes comparison

3rd model

For this model were able to achieve in 100 epochs:

test/accuracy:0.7711997641741181
test/f1:0.8607231510003888
test/precision:0.8100653006079712
test/roc_auc:0.7224563738997474
test/sensitivity:0.9181394755311684
test/specificity:0.2792138431958983
train/loss:0.2656220795832715

This model has high sensitivity.

Confusion matrix:

Confusion Matrix

We have used this configuration file:

n_input: 47 layers: [90, 29, 1]
activations: [ReLU, ReLU, null]
output_activation: null
dropout: 0.2
batch_norm: true
skip_connections: false
bottleneck: true

training:
  epochs: 100
  lr: 0.01
  batch_size: 128
  verbose: true
  patience: 10
  min_delta: 0.001

Process of training:

Learning processes comparison