РК1 - ТМО. Вариант 12

номер датасета - 4 номер задачи - 2 Для заданного набора данных проведите обработку пропусков в данных для одного категориального и одного количественного признака. Какие способы обработки пропусков в данных для категориальных и количественных признаков Вы использовали? Какие признаки Вы будете использовать для дальнейшего построения моделей машинного обучения и почему?

Основные атрибуты:

1. age — возраст
2. sex — пол
3. chest pain type — тип боли в груди (4 значения)
4. resting blood pressure — артериальное давление в состоянии покоя
5. serum cholestoral in mg/dl — уровень холестерина в сыворотке крови (мг/дл)
6. fasting blood sugar > 120 mg/dl — уровень сахара в крови натощак > 120 мг/дл
7. resting electrocardiographic results — результаты ЭКГ в состоянии покоя (значения 0, 1, 2)
8. maximum heart rate achieved — максимальная достигнутая частота сердечных сокращений
9. exercise induced angina — стенокардия, вызванная физической нагрузкой
10. oldpeak — депрессия ST сегмента, вызванная физической нагрузкой относительно состояния покоя
11. the slope of the peak exercise ST segment — наклон пикового ST сегмента при физической нагрузке
12. number of major vessels (0-3) colored by flourosopy — количество крупных сосудов (0-3), окрашенных флюороскопией
13. thal — состояние талия: 0 = нормальное 1 = фиксированный дефект 2 = обратимый дефект

Загрузка датасета

```
pip install kaggle
```

```
Collecting kaggle
  Downloading kaggle-1.7.4.2-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: bleach in c:\users\lerum\anaconda3\lib\
site-packages (from kaggle) (6.2.0)
Requirement already satisfied: certifi>=14.05.14 in c:\users\lerum\
anaconda3\lib\site-packages (from kaggle) (2025.1.31)
Requirement already satisfied: charset-normalizer in c:\users\lerum\
anaconda3\lib\site-packages (from kaggle) (3.3.2)
Requirement already satisfied: idna in c:\users\lerum\anaconda3\lib\
site-packages (from kaggle) (3.7)
Collecting protobuf (from kaggle)
  Downloading protobuf-6.30.2-cp39-cp39-win_amd64.whl.metadata (593
bytes)
```

```
Requirement already satisfied: python-dateutil>=2.5.3 in c:\users\
lerum\anaconda3\lib\site-packages (from kaggle) (2.9.0.post0)
Requirement already satisfied: python-slugify in c:\users\lerum\
anaconda3\lib\site-packages (from kaggle) (5.0.2)
Requirement already satisfied: requests in c:\users\lerum\anaconda3\
lib\site-packages (from kaggle) (2.32.3)
Requirement already satisfied: setuptools>=21.0.0 in c:\users\lerum\
anaconda3\lib\site-packages (from kaggle) (75.8.0)
Requirement already satisfied: six>=1.10 in c:\users\lerum\anaconda3\
lib\site-packages (from kaggle) (1.16.0)
Requirement already satisfied: text-unidecode in c:\users\lerum\
anaconda3\lib\site-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in c:\users\lerum\anaconda3\lib\
site-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in c:\users\lerum\
anaconda3\lib\site-packages (from kaggle) (2.3.0)
Requirement already satisfied: webencodings in c:\users\lerum\
anaconda3\lib\site-packages (from kaggle) (0.5.1)
Requirement already satisfied: colorama in c:\users\lerum\anaconda3\
lib\site-packages (from tqdm->kaggle) (0.4.6)
Downloading kaggle-1.7.4.2-py3-none-any.whl (173 kB)
Downloading protobuf-6.30.2-cp39-cp39-win_amd64.whl (431 kB)
Installing collected packages: protobuf, kaggle
Successfully installed kaggle-1.7.4.2 protobuf-6.30.2
Note: you may need to restart the kernel to use updated packages.
```

```
import kaggle
kaggle.api.dataset_download_files("johnsmith88/heart-disease-dataset",
path="datasets", unzip=True)
```

```
Dataset URL: https://www.kaggle.com/datasets/johnsmith88/heart-
disease-dataset
```

```
import pandas as pd

df = pd.read_csv("datasets/heart.csv")
df
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang
oldpeak  \
0      52    1   0       125   212    0        1      168      0
1.0
1      53    1   0       140   203    1        0      155      1
3.1
2      70    1   0       145   174    0        1      125      1
2.6
3      61    1   0       148   203    0        1      161      0
0.0
4      62    0   0       138   294    1        1      106      0
1.9
```

```
...   ...  ...  ..       ...   ...  ...      ...      ...    ...
...
1020   59    1   1       140   221    0        1      164      1
0.0
1021   60    1   0       125   258    0        0      141      1
2.8
1022   47    1   0       110   275    0        0      118      1
1.0
1023   50    0   0       110   254    0        0      159      0
0.0
1024   54    1   0       120   188    0        1      113      0
1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
3         2   1     3       0
4         1   3     2       0
...     ...  ..   ...     ...
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[1025 rows x 14 columns]
```

Создание категориального признака

```
df["chol_category"] = pd.cut(df["chol"], bins=[0, 200, 239,
float("inf")], labels=["низкий", "средний", "высокий"])

df = df.drop('chol', axis = 1)
df
```

```
      age  sex  cp  trestbps  fbs  restecg  thalach  exang  oldpeak
slope  \
0      52    1   0       125    0        1      168      0      1.0
2
1      53    1   0       140    1        0      155      1      3.1
0
2      70    1   0       145    0        1      125      1      2.6
0
3      61    1   0       148    0        1      161      0      0.0
2
4      62    0   0       138    1        1      106      0      1.9
1
...   ...  ...  ..       ...  ...      ...      ...    ...      ...
...
```

```
1020   59    1   1       140    0        1      164      1      0.0
2
1021   60    1   0       125    0        0      141      1      2.8
1
1022   47    1   0       110    0        0      118      1      1.0
1
1023   50    0   0       110    0        0      159      0      0.0
2
1024   54    1   0       120    0        1      113      0      1.4
1
```

```
      ca  thal  target chol_category
0      2     3       0       средний
1      0     3       0       средний
2      0     3       0        низкий
3      1     3       0       средний
4      3     2       0       высокий
...   ..   ...     ...           ...
1020   0     2       1       средний
1021   1     3       0       высокий
1022   1     2       0       высокий
1023   0     2       1       высокий
1024   1     3       0        низкий

[1025 rows x 14 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   age            1025 non-null   int64
 1   sex            1025 non-null   int64
 2   cp             1025 non-null   int64
 3   trestbps       1025 non-null   int64
 4   fbs            1025 non-null   int64
 5   restecg        1025 non-null   int64
 6   thalach        1025 non-null   int64
 7   exang          1025 non-null   int64
 8   oldpeak        1025 non-null   float64
 9   slope          1025 non-null   int64
 10  ca             1025 non-null   int64
 11  thal           1025 non-null   int64
 12  target         1025 non-null   int64
 13  chol_category  1025 non-null   category
dtypes: category(1), float64(1), int64(12)
memory usage: 105.4 KB
```

Внесение пропусков в возраст и chol_category

```
import numpy as np

def introduce_nan(data, missing_rate=0.1, columns=['age',
'chol_category']):
    df_copy = data.copy()
    for col in columns:
        if col in df_copy.columns:
            mask = np.random.rand(len(df_copy)) < missing_rate
            df_copy.loc[mask, col] = np.nan
    return df_copy

df_missing = introduce_nan(df, missing_rate=0.15, columns=['age',
'chol_category'])

df_missing
```

```
       age  sex  cp  trestbps  fbs  restecg  thalach  exang  oldpeak
slope  \
0     52.0    1   0       125    0        1      168      0      1.0
2
1      NaN    1   0       140    1        0      155      1      3.1
0
2     70.0    1   0       145    0        1      125      1      2.6
0
3     61.0    1   0       148    0        1      161      0      0.0
2
4     62.0    0   0       138    1        1      106      0      1.9
1
...    ...  ...  ..       ...  ...      ...      ...    ...      ...
...
1020  59.0    1   1       140    0        1      164      1      0.0
2
1021  60.0    1   0       125    0        0      141      1      2.8
1
1022  47.0    1   0       110    0        0      118      1      1.0
1
1023  50.0    0   0       110    0        0      159      0      0.0
2
1024  54.0    1   0       120    0        1      113      0      1.4
1

      ca  thal  target chol_category
0      2     3       0       средний
1      0     3       0       средний
2      0     3       0           NaN
3      1     3       0       средний
4      3     2       0       высокий
...   ..   ...     ...           ...
```

```
1020   0     2        1       средний
1021   1     3        0       высокий
1022   1     2        0       высокий
1023   0     2        1       высокий
1024   1     3        0        низкий

[1025 rows x 14 columns]
```

```
df_missing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype   
---  ------         --------------  -----   
 0   age            867 non-null    float64 
 1   sex            1025 non-null   int64   
 2   cp             1025 non-null   int64   
 3   trestbps       1025 non-null   int64   
 4   fbs            1025 non-null   int64   
 5   restecg        1025 non-null   int64   
 6   thalach        1025 non-null   int64   
 7   exang          1025 non-null   int64   
 8   oldpeak        1025 non-null   float64 
 9   slope          1025 non-null   int64   
 10  ca             1025 non-null   int64   
 11  thal           1025 non-null   int64   
 12  target         1025 non-null   int64   
 13  chol_category  881 non-null    category
dtypes: category(1), float64(2), int64(11)
memory usage: 105.4 KB
```

Заполнение пропусков для категориального признака

я выполню упрощенный вариант, заполню все пропуски в количественных признаках медианой через simpleImputer.Медианой буду заполнять так как она устойчива к выбросам. Заполнять категориальные признаки буду с помощью модели KNN. Но перед обучением модели сначала разделю выборку на трейн и тест, причем к тесту отнесу все значения, где пропущен признак chol_category

```
num_x = df_missing.drop(df_missing.columns[-1], axis=1) # извлекаем количественные признаки
cat_x = df_missing[df_missing.columns[-1]]#извлекаем категориальные признаки
```

```
from sklearn.impute import SimpleImputer
imputer_num = SimpleImputer(strategy="median")
num_x_impute = imputer_num.fit_transform(num_x)
```

- через SimpleImputer заполнила пропуски количественных признаков медианами

```
df_prep = pd.DataFrame(num_x_impute, columns = num_x.columns)
df_prep['chol_category'] = cat_x
df_prep
```

```
       age  sex   cp  trestbps  fbs  restecg  thalach  exang  oldpeak
slope  \
0     52.0  1.0  0.0     125.0  0.0      1.0    168.0    0.0      1.0
2.0
1     56.0  1.0  0.0     140.0  1.0      0.0    155.0    1.0      3.1
0.0
2     70.0  1.0  0.0     145.0  0.0      1.0    125.0    1.0      2.6
0.0
3     61.0  1.0  0.0     148.0  0.0      1.0    161.0    0.0      0.0
2.0
4     62.0  0.0  0.0     138.0  1.0      1.0    106.0    0.0      1.9
1.0
...    ...  ...  ...       ...  ...      ...      ...    ...      ...
...
1020  59.0  1.0  1.0     140.0  0.0      1.0    164.0    1.0      0.0
2.0
1021  60.0  1.0  0.0     125.0  0.0      0.0    141.0    1.0      2.8
1.0
1022  47.0  1.0  0.0     110.0  0.0      0.0    118.0    1.0      1.0
1.0
1023  50.0  0.0  0.0     110.0  0.0      0.0    159.0    0.0      0.0
2.0
1024  54.0  1.0  0.0     120.0  0.0      1.0    113.0    0.0      1.4
1.0

       ca  thal  target chol_category
0     2.0   3.0     0.0       средний
1     0.0   3.0     0.0       средний
2     0.0   3.0     0.0           NaN
3     1.0   3.0     0.0       средний
4     3.0   2.0     0.0       высокий
...   ...   ...     ...           ...
1020  0.0   2.0     1.0       средний
1021  1.0   3.0     0.0       высокий
1022  1.0   2.0     0.0       высокий
1023  0.0   2.0     1.0       высокий
1024  1.0   3.0     0.0        низкий

[1025 rows x 14 columns]
```

```
df_prep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype   
---  ------         --------------  -----   
 0   age            1025 non-null   float64 
 1   sex            1025 non-null   float64 
 2   cp             1025 non-null   float64 
 3   trestbps       1025 non-null   float64 
 4   fbs            1025 non-null   float64 
 5   restecg        1025 non-null   float64 
 6   thalach        1025 non-null   float64 
 7   exang          1025 non-null   float64 
 8   oldpeak        1025 non-null   float64 
 9   slope          1025 non-null   float64 
 10  ca             1025 non-null   float64 
 11  thal           1025 non-null   float64 
 12  target         1025 non-null   float64 
 13  chol_category  881 non-null    category
dtypes: category(1), float64(13)
memory usage: 105.4 KB
```

```
from sklearn.preprocessing import LabelEncoder
df_prep['chol_category'] =
df_prep['chol_category'].astype('object')#преобразуем категориальный
признак, в тип объект, иначе ругается(
le = LabelEncoder()#кодируем категории
chol_col = df_prep['chol_category']
not_null_mask = chol_col.notna()#создаем маску, оставляем только
элементы, которые не равны Nan
df_prep.loc[not_null_mask, 'chol_category'] =
le.fit_transform(chol_col[not_null_mask])#кодируем только те строки
категориального признака, которые попадают под маску
df_prep['chol_category'] =
df_prep['chol_category'].astype('category')#преобразуем тип признака
обратно в категориальный
```

```
df_prep
```

```
       age  sex   cp  trestbps  fbs  restecg  thalach  exang  oldpeak
slope  \
0     52.0  1.0  0.0     125.0  0.0      1.0    168.0    0.0      1.0
2.0
1     56.0  1.0  0.0     140.0  1.0      0.0    155.0    1.0      3.1
0.0
2     70.0  1.0  0.0     145.0  0.0      1.0    125.0    1.0      2.6
0.0
3     61.0  1.0  0.0     148.0  0.0      1.0    161.0    0.0      0.0
```

```
2.0
4     62.0  0.0  0.0     138.0  1.0      1.0    106.0    0.0      1.9
1.0
...    ...  ...  ...       ...  ...      ...      ...    ...      ...
...
1020  59.0  1.0  1.0     140.0  0.0      1.0    164.0    1.0      0.0
2.0
1021  60.0  1.0  0.0     125.0  0.0      0.0    141.0    1.0      2.8
1.0
1022  47.0  1.0  0.0     110.0  0.0      0.0    118.0    1.0      1.0
1.0
1023  50.0  0.0  0.0     110.0  0.0      0.0    159.0    0.0      0.0
2.0
1024  54.0  1.0  0.0     120.0  0.0      1.0    113.0    0.0      1.4
1.0

       ca  thal  target chol_category
0     2.0   3.0     0.0             2
1     0.0   3.0     0.0             2
2     0.0   3.0     0.0           NaN
3     1.0   3.0     0.0             2
4     3.0   2.0     0.0             0
...   ...   ...     ...           ...
1020  0.0   2.0     1.0             2
1021  1.0   3.0     0.0             0
1022  1.0   2.0     0.0             0
1023  0.0   2.0     1.0             0
1024  1.0   3.0     0.0             1

[1025 rows x 14 columns]

df_prep.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   age            1025 non-null   float64
 1   sex            1025 non-null   float64
 2   cp             1025 non-null   float64
 3   trestbps       1025 non-null   float64
 4   fbs            1025 non-null   float64
 5   restecg        1025 non-null   float64
 6   thalach        1025 non-null   float64
 7   exang          1025 non-null   float64
 8   oldpeak        1025 non-null   float64
 9   slope          1025 non-null   float64
 10  ca             1025 non-null   float64
 11  thal           1025 non-null   float64
```

```
 12  target         1025 non-null   float64
 13  chol_category  881 non-null    category
dtypes: category(1), float64(13)
memory usage: 105.4 KB
```

Разделим выборку на трейн и тест

```
train_df = df_prep[df_prep['chol_category'].notna()].copy()
test_df = df_prep[df_prep['chol_category'].isna()].copy()

test_df_X = test_df.drop(['chol_category', 'target'], axis=1)

test_df_X
```

```
      age  sex   cp  trestbps  fbs  restecg  thalach  exang  oldpeak
slope  \
2    70.0  1.0  0.0     145.0  0.0      1.0    125.0    1.0      2.6
0.0
10   71.0  0.0  0.0     112.0  0.0      1.0    125.0    0.0      1.6
1.0
36   51.0  1.0  3.0     125.0  0.0      0.0    125.0    1.0      1.4
2.0
48   66.0  0.0  2.0     146.0  0.0      0.0    152.0    0.0      0.0
1.0
50   58.0  0.0  3.0     150.0  1.0      0.0    162.0    0.0      1.0
2.0
..    ...  ...  ...       ...  ...      ...      ...    ...      ...
...
989  71.0  0.0  1.0     160.0  0.0      1.0    162.0    0.0      0.4
2.0
991  60.0  1.0  0.0     117.0  1.0      1.0    160.0    1.0      1.4
2.0
992  50.0  0.0  0.0     110.0  0.0      0.0    159.0    0.0      0.0
2.0
993  43.0  1.0  0.0     132.0  1.0      0.0    143.0    1.0      0.1
1.0
995  44.0  1.0  1.0     120.0  0.0      1.0    173.0    0.0      0.0
2.0

      ca  thal
2    0.0   3.0
10   0.0   2.0
36   1.0   2.0
48   1.0   2.0
50   0.0   2.0
..   ...   ...
989  2.0   2.0
991  2.0   3.0
992  0.0   2.0
993  4.0   3.0
```

```
995  0.0   3.0

[144 rows x 12 columns]
```

```
train_df_X = train_df.drop(['chol_category', 'target'], axis=1)
train_df_y = train_df[train_df.columns[-1]]
```

Нормализуем количественные признаки

```
from sklearn.preprocessing import MinMaxScaler
```

```
train_df_y
```

```
0       2
1       2
3       2
4       0
5       0
       ..
1020    2
1021    0
1022    0
1023    0
1024    1
Name: chol_category, Length: 881, dtype: category
Categories (3, int64): [0, 1, 2]
```

```
scaler = MinMaxScaler()
train_df_X_scaled = scaler.fit_transform(train_df_X)
test_df_X_scaled = scaler.transform(test_df_X)
```

Обучаем модель KNN

и с помощью грид серча подбираем подходящие параметры

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
param_grid = {
    'n_neighbors':  list(range(1, 100))
}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(knn, param_grid, cv=cv, scoring='accuracy')
grid_search.fit(train_df_X_scaled, train_df_y)
```

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=42,
shuffle=True),
             estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9,
```

```
10, 11, 12,
                                       13, 14, 15, 16, 17, 18, 19,
20, 21, 22,
                                       23, 24, 25, 26, 27, 28, 29,
30, ...]},
             scoring='accuracy')
```

```
knn_best_param = KNeighborsClassifier(n_neighbors=1)
knn_best_param.fit(train_df_X_scaled, train_df_y)
```

```
KNeighborsClassifier(n_neighbors=1)
```

Предсказываем на тесте:

```
predicted_chol = knn_best_param.predict(test_df_X_scaled)
```

Заменяем пропущенные признаки на предсказанные в исходном датасете

```
df_prep.loc[df_prep['chol_category'].isna(), 'chol_category'] =
predicted_chol
```

```
df_prep
```

```
       age  sex   cp  trestbps  fbs  restecg  thalach  exang  oldpeak
slope  \
0     52.0  1.0  0.0     125.0  0.0      1.0    168.0    0.0      1.0
2.0
1     56.0  1.0  0.0     140.0  1.0      0.0    155.0    1.0      3.1
0.0
2     70.0  1.0  0.0     145.0  0.0      1.0    125.0    1.0      2.6
0.0
3     61.0  1.0  0.0     148.0  0.0      1.0    161.0    0.0      0.0
2.0
4     62.0  0.0  0.0     138.0  1.0      1.0    106.0    0.0      1.9
1.0
...    ...  ...  ...       ...  ...      ...      ...    ...      ...
...
1020  59.0  1.0  1.0     140.0  0.0      1.0    164.0    1.0      0.0
2.0
1021  60.0  1.0  0.0     125.0  0.0      0.0    141.0    1.0      2.8
1.0
1022  47.0  1.0  0.0     110.0  0.0      0.0    118.0    1.0      1.0
1.0
1023  50.0  0.0  0.0     110.0  0.0      0.0    159.0    0.0      0.0
2.0
1024  54.0  1.0  0.0     120.0  0.0      1.0    113.0    0.0      1.4
1.0
```

```
       ca  thal  target chol_category
0     2.0   3.0     0.0             2
1     0.0   3.0     0.0             2
2     0.0   3.0     0.0             1
3     1.0   3.0     0.0             2
4     3.0   2.0     0.0             0
...   ...   ...     ...           ...
1020  0.0   2.0     1.0             2
1021  1.0   3.0     0.0             0
1022  1.0   2.0     0.0             0
1023  0.0   2.0     1.0             0
1024  1.0   3.0     0.0             1

[1025 rows x 14 columns]
```

```
df_prep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype   
---  ------         --------------  -----   
 0   age            1025 non-null   float64 
 1   sex            1025 non-null   float64 
 2   cp             1025 non-null   float64 
 3   trestbps       1025 non-null   float64 
 4   fbs            1025 non-null   float64 
 5   restecg        1025 non-null   float64 
 6   thalach        1025 non-null   float64 
 7   exang          1025 non-null   float64 
 8   oldpeak        1025 non-null   float64 
 9   slope          1025 non-null   float64 
 10  ca             1025 non-null   float64 
 11  thal           1025 non-null   float64 
 12  target         1025 non-null   float64 
 13  chol_category  1025 non-null   category
dtypes: category(1), float64(13)
memory usage: 105.4 KB
```

Проверка качества классификатора на трейне с помощью кросс валидации

```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(knn_best_param, train_df_X_scaled,
train_df_y, cv=5)
cv_scores.mean()
```

```
0.9352786337955828
```

Jointplot для age и trestbps

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.jointplot(x='age', y='trestbps', data=df, kind='scatter')
plt.show()
```