

Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



Лабораторная работа № 2

по дисциплине «Методы машинного обучения»

Обработка признаков, часть 1

ИСПОЛНИТЕЛЬ:

студент ИУ5-23М

Бондаренко И. Г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю. Е.

___ " _____ " 2024 г.

Москва, 2024

✓ Задание лабораторной работы

- Выбрать набор данных (датасет), содержащий категориальные и числовые признаки и пропуски в данных. Для выполнения следующих пунктов можно использовать несколько различных наборов данных (один для обработки пропусков, другой для категориальных признаков и т.д.) Просьба не использовать датасет, на котором данная задача решалась в лекции.
- Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи:
 - устранение пропусков в данных;
 - кодирование категориальных признаков;
 - нормализация числовых признаков.


✓ Выполнение работы

✓ Импорт библиотек

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from IPython.display import Image
%matplotlib inline
sns.set(style="ticks")
```

✓ Подключение Google Диска для работы с Google Colab


```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

✓ Чтение данных

```
data = pd.read_csv('/content/drive/MyDrive/MMO/PopularSpotifySongs.csv', encoding='unicode_escape')
```

```
data.head()
```



	track_name	artist(s)_name	artist_count	released_year	released_month	released_
0	Seven (feat. Latto) (Explicit Ver.)	Latto, Jung Kook	2	2023	7	
1	LALA	Myke Towers	1	2023	3	
2	vampire	Olivia Rodrigo	1	2023	6	
3	Cruel Summer	Taylor Swift	1	2019	8	
4	WHERE SHE GOES	Bad Bunny	1	2023	5	

5 rows × 24 columns

```
data.isnull().sum()
```

```

track_name          0
artist(s)_name      0
artist_count        0
released_year        0
released_month       0
released_day         0
in_spotify_playlists 0
in_spotify_charts    0
streams             0
in_apple_playlists  0
in_apple_charts     0
in_deezer_playlists 0
in_deezer_charts    0
in_shazam_charts    50
bpm                 0
key                 95
mode                0
danceability_%       0
valence_%            0
energy_%             0
acousticness_%       0
instrumentalness_%   0
liveness_%           0
speechiness_%        0
dtype: int64

```

✓ Устранение пропусков

Определим столбцы, в которых наблюдаются пропуски данных:

```

for column in data.columns:
    if (data[column].isnull().sum() != 0):
        print(column, ': ', data[column].isnull().sum())

```

```

in_shazam_charts : 50
key : 95

```

В столбцах bodyType, enginePower, modelDate, vehicleTransmission, Владелец, ПТС, Привод, price, price_EUR, price_USD существуют строки, содержащие пропуски данных, их необходимо удалить.

Удалим пропуски в bodyType:

```
data.drop(data[data['in_shazam_charts'].isnull()].index, inplace=True)
```

Проверим снова:

```

for column in data.columns:
    if (data[column].isnull().sum() != 0):
        print(column, ': ', data[column].isnull().sum())

```

```
key : 86
```

Видим, что число столбцов сократилось. Из этого следует, что удаленные строки содержали пропуски данных в нескольких столбцах. Удалим еще строки с пропусками:

```
data.drop(data[data['key'].isnull()].index, inplace=True)
```

Убедимся с помощью процентного соотношения, что пропусков в столбцах нет:

```

for col in data.columns:
    pct_missing = np.mean(data[col].isnull())
    print('{} - {}'.format(col, round(pct_missing*100)))

```

```

track_name - 0%
artist(s)_name - 0%
artist_count - 0%
released_year - 0%
released_month - 0%
released_day - 0%
in_spotify_playlists - 0%
in_spotify_charts - 0%
streams - 0%
in_apple_playlists - 0%
in_apple_charts - 0%

```

```

in_deezer_playlists - 0%
in_deezer_charts - 0%
in_shazam_charts - 0%
bpm - 0%
key - 0%
mode - 0%
danceability_% - 0%
valence_% - 0%
energy_% - 0%
acousticness_% - 0%
instrumentalness_% - 0%
liveness_% - 0%
speechiness_% - 0%

```

✓ Кодирование категориальных признаков

✓ LabelEncoder

Выберем два категориальных признака - `key` и `mode`. Их закодируем с помощью `LabelEncoder`.

```
data['mode'].unique()
```

```
→ array(['Major', 'Minor'], dtype=object)
```

```
data['key'].unique()
```

```
→ array(['B', 'C#', 'F', 'A', 'D', 'F#', 'G#', 'G', 'E', 'A#', 'D#'],
      dtype=object)
```

Закодируем их в числовые значения:

```
from sklearn.preprocessing import LabelEncoder
```

```

lekey = LabelEncoder()
lekey.fit_transform(data["key"])
data["key"] = lekey.transform(data["key"])
data = data.astype({"key": "int64"})

```

```

lemode = LabelEncoder()
lemode.fit_transform(data["mode"])
data["mode"] = lemode.transform(data["mode"])
data = data.astype({"mode": "int64"})

```

```
data['mode'].unique()
```

```
→ array([0, 1])
```

```
data['key'].unique()
```

```
→ array([ 2,  3,  7,  0,  4,  8, 10,  9,  6,  1,  5])
```

✓ OneHotEncoder

Для признака `fuelType` проведем кодирование бинарными значениями с помощью `OneHotEncoder`.

```
data['key'].unique()
```

```
→ array(['B', 'C#', 'F', 'A', 'D', 'F#', nan, 'G#', 'G', 'E', 'A#', 'D#'],
      dtype=object)
```

```
from sklearn.preprocessing import OneHotEncoder
```

```

ohe = OneHotEncoder()
cat_enc_ohe = ohe.fit_transform(data[['key']])
cat_enc_ohe

```

```
→ <953x12 sparse matrix of type '<class 'numpy.float64'>'
   with 953 stored elements in Compressed Sparse Row format>
```

```
cat_enc_ohe.todense()[0:10]
```

```
matrix([[0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]])
```

```
pd.get_dummies(data[['key']]).head()
```

	key_A	key_A#	key_B	key_C#	key_D	key_D#	key_E	key_F	key_F#	key_G	key_G#
0	False	False	True	False	False	False	False	False	False	False	False
1	False	False	False	True	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	True	False	False	False
3	True	False	False	False	False	False	False	False	False	False	False
4	True	False	False	False	False	False	False	False	False	False	False

```
pd.get_dummies(data[['key']], dummy_na=True).head()
```

	key_A	key_A#	key_B	key_C#	key_D	key_D#	key_E	key_F	key_F#	key_G	key_G#	k
0	False	False	True	False	False	False	False	False	False	False	False	
1	False	False	False	True	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	True	False	False	False	
3	True	False	False	False	False	False	False	False	False	False	False	

CountEncoder

Для кодирования key используем CountEncoder.

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
    81.9/81.9 kB 2.3 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.0.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2021.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2022.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (2.0.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encoders) (21.3)
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```
from category_encoders.count import CountEncoder as ce_CountEncoder
```

```
data['key'].unique()
```

```
array(['B', 'C#', 'F', 'A', 'D', 'F#', nan, 'G#', 'G', 'E', 'A#', 'D#'],
      dtype=object)
```

```
ce_CountEncoder1 = ce_CountEncoder()
data_COUNT_ENC = ce_CountEncoder1.fit_transform(data['key'])
```

```
data_COUNT_ENC['key'].unique()
```

```
array([ 81, 120, 89, 75, 73, 95, 91, 96, 62, 57, 33])
```

✓ FrequencyEncoder

Для признака artist_count используем FrequencyEncoder.

```
data['key'].unique()

array(['B', 'C#', 'F', 'A', 'D', 'F#', nan, 'G#', 'G', 'E', 'A#', 'D#'],
      dtype=object)

ce_CountEncoder2 = ce_CountEncoder(normalize=True)
data_FREQ_ENC = ce_CountEncoder2.fit_transform(data['key'])

data_FREQ_ENC['key'].unique()

array([0.08499475, 0.12591815, 0.0933893 , 0.07869885, 0.07660021,
       0.0996852 , 0.09548793, 0.10073452, 0.06505771, 0.05981112,
       0.03462749])
```

✓ Нормализация числовых признаков

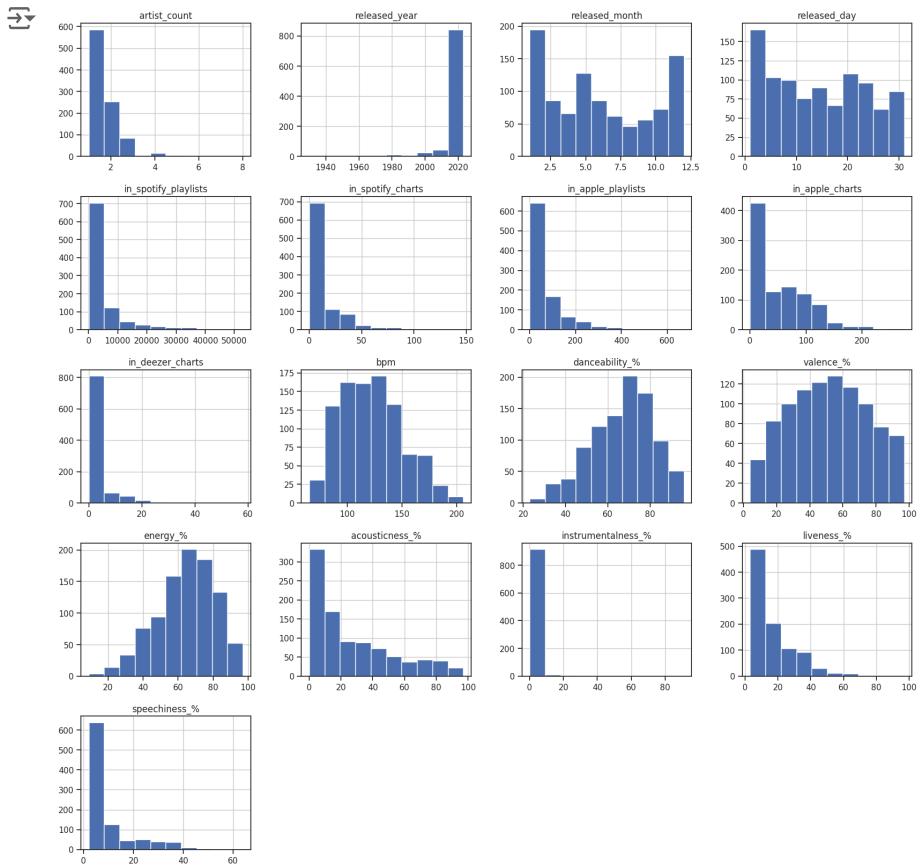
Нормализация числового признака предполагает что на основе существующего признака мы создаем новый признак, который в идеале имеет нормальное распределение.

```
def diagnostic_plots(df, variable):
    plt.figure(figsize=(15,6))
    # гистограмма
    plt.subplot(1, 2, 1)
    df[variable].hist(bins=30)
    ## Q-Q plot
    plt.subplot(1, 2, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.show()

data.hist(figsize=(20,20))

def cock(s):
    return s + 1

data['acousticness_%'] = data['acousticness_%'].apply(cock)
plt.show()
```



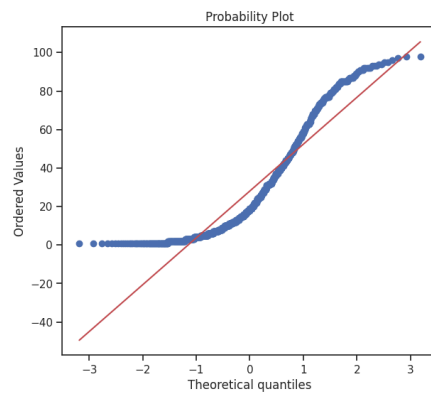
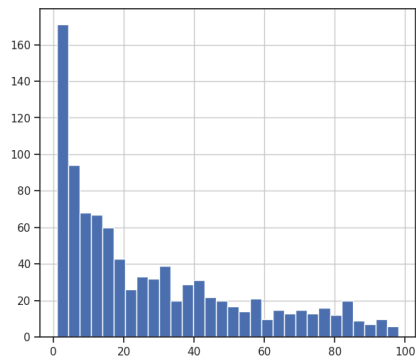
```
data['acousticness_%'].unique()
```

```
array([32,  8, 18, 12, 15, 20, 49, 38, 13, 22, 24, 19,  7, 84, 35,  6, 44,
       97,  1, 23, 27, 10, 47,  3, 40, 37, 68, 28, 55,  5, 53, 11, 17, 71,
       52, 59, 60, 56, 51, 26, 63, 75,  2, 45, 66, 34, 79, 77,  4, 76, 16,
       72, 62, 31, 25, 54, 94, 33, 46, 43, 39,  9, 74, 58, 98, 95, 21, 87,
       64, 85, 41, 29, 82, 42, 70, 61, 90, 67, 36, 73, 48, 80, 50, 81, 92,
       57, 30, 14, 91, 78, 88, 86, 65, 89, 69, 96, 93, 83])
```

✓ Исходное распределение

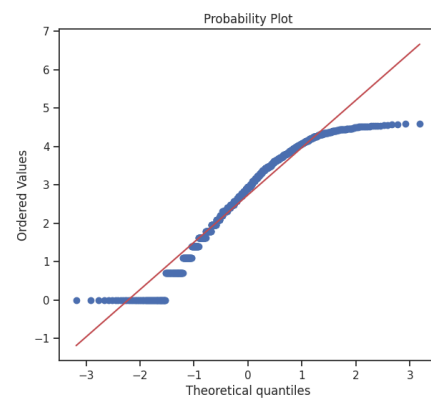
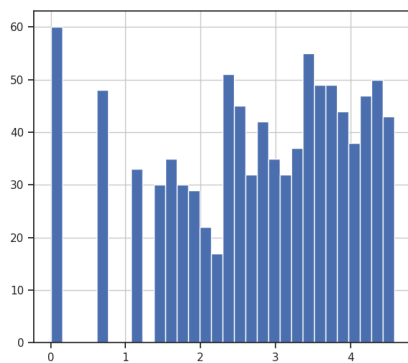
Исходное распределение для признака числового признака `enginePower`:

```
diagnostic_plots(data, 'acousticness_%')
```



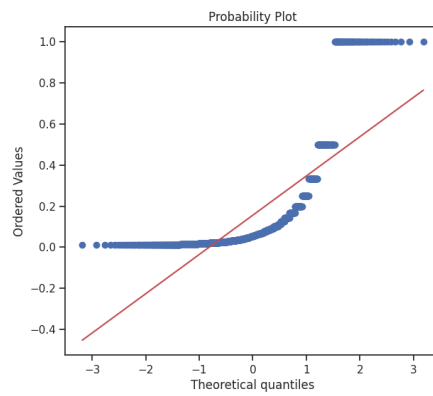
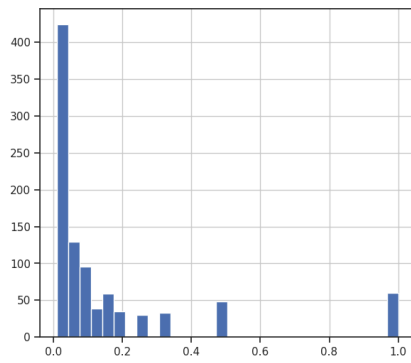
✓ Логарифмическое преобразование

```
data['acousticness_%_log'] = np.log(data['acousticness_%'])  
diagnostic_plots(data, 'acousticness_%_log')
```



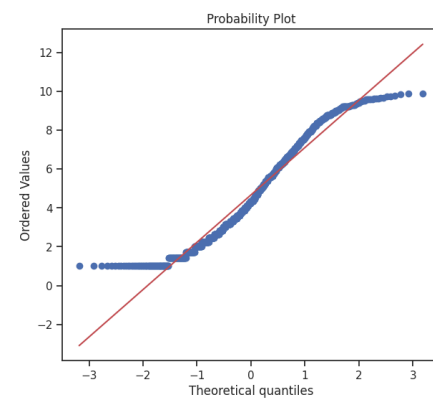
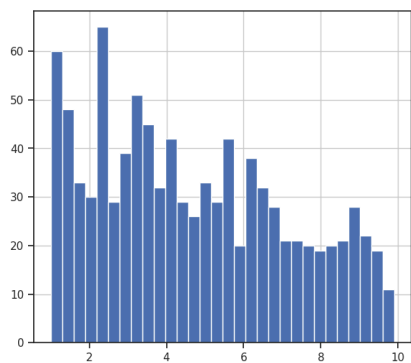
✓ Обратное преобразование

```
data['acousticness_%_reciprocal'] = 1 / (data['acousticness_%'])  
diagnostic_plots(data, 'acousticness_%_reciprocal')
```

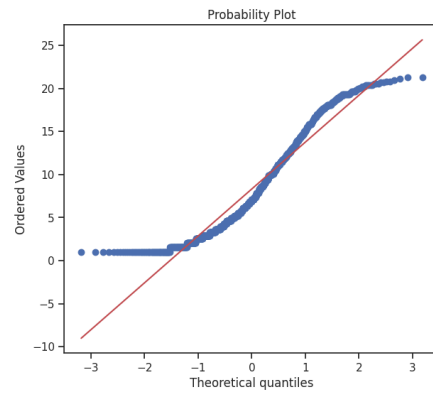
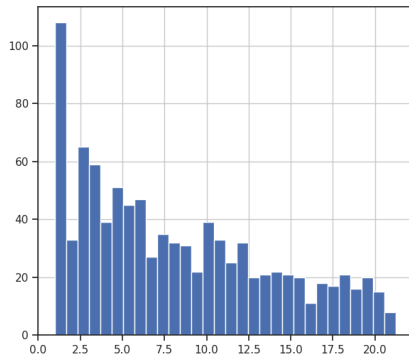
✓ Преобразование с использованием квадратного корня

```
data['acousticness_%_sqr'] = data['acousticness_%']**(1/2)
diagnostic_plots(data, 'acousticness_%_sqr')
```



✓ Возведение в степень

```
data['acousticness_%_exp'] = data['acousticness_%']**(1/1.5)
diagnostic_plots(data, 'acousticness_%_exp')
```

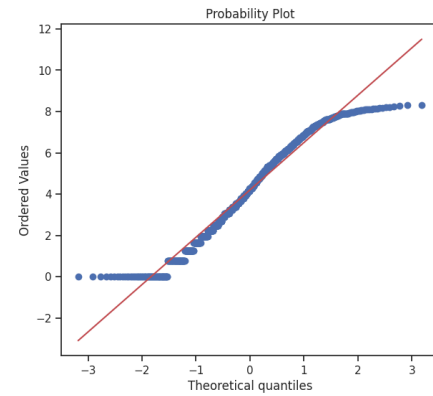
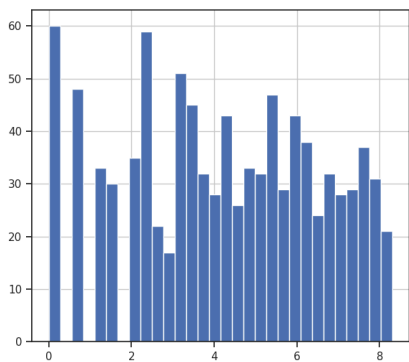


✓ Преобразование Бокса-Кокса

```
data['acousticness_%_cox'], param = stats.boxcox(data['acousticness_%'])
print('Оптимальное значение  $\lambda = \{ }\'.format(param))
diagnostic_plots(data, 'acousticness_%_cox')$ 
```



Оптимальное значение $\lambda = 0.23762579788908522$



✓ Преобразование Йео-Джонсона

```
# Необходимо преобразовать данные к действительному типу
data['acousticness_%'] = data['acousticness_%'].astype('float')
data['acousticness_%_yeojohnson'], param = stats.yeojohnson(data['acousticness_%'])
print('Оптимальное значение  $\lambda = \{ }\'.format(param))
diagnostic_plots(data, 'acousticness_%_yeojohnson')$ 
```