Гаврилов Л.Я. ИУ5-23М

Обработка признаков (часть 1).

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

Загрузка и первичный анализ данных

Используем датасет [PC Games 2020] игр Steam с добавлением данных RAWG API

```
data = pd.read_csv('games.csv', sep=",")
```

```
data.shape
```

```
(30250, 27)
```

```
data.dtypes
```

```
Unnamed: 0           int64
id                   int64
Name                object
RawgID             float64
SteamURL            object
Metacritic         float64
Genres              object
Indie              float64
Presence           float64
Platform            object
Graphics            object
Storage             object
Memory              object
RatingsBreakdown    object
ReleaseDate         object
Soundtrack         float64
Franchise           object
OriginalCost        object
DiscountedCost      object
Players             object
Controller         float64
Languages           object
ESRB                object
Achievements       float64
Publisher          float64
Description         object
Tags                object
dtype: object
```

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

```
Unnamed: 0              0
id                      0
Name                   94
RawgID                 94
SteamURL               55
Metacritic          26894
Genres               2968
Indie                 205
Presence               94
Platform              127
Graphics             4320
Storage              2759
Memory               1934
RatingsBreakdown    15206
ReleaseDate          3226
Soundtrack            205
Franchise           25163
OriginalCost          746
DiscountedCost      29523
Players             17916
Controller            274
Languages             223
```

```
ESRB              25503
Achievements         94
Publisher         30250
Description         219
Tags                205
dtype: int64
```

```
data.head()
```

	Unnamed: 0	id	Name	RawgID	SteamURL	Metacritic	Genres	Indie	Presence	Platform	...
0	0	1	Counter-Strike: Global Offensive	4291.0	https://store.steampowered.com/app/730/?snr=1_...	83.0	Action, Free to Play	0.0	1009588.0	PC, Xbox 360, PlayStation 3	...
1	1	2	Destiny 2	32.0	https://store.steampowered.com/app/1085660/?sn...	82.0	Action, Adventure, Free to Play	0.0	1007425.0	PlayStation 5, Web, Xbox Series X, PC, Xbox On...	...
2	2	3	Dota 2	10213.0	https://store.steampowered.com/app/570/?snr=1_...	90.0	NaN	0.0	1009306.0	Linux, macOS, PC	...
3	3	4	The Elder Scrolls Online	41458.0	https://store.steampowered.com/app/306130/?snr...	71.0	Massively Multiplayer, RPG	0.0	1000781.0	PC	...
4	4	5	Sea of Thieves	50781.0	https://store.steampowered.com/app/1172620/?sn...	68.0	Action, Adventure	0.0	777456.0	PC, Xbox One	...

5 rows × 27 columns

```
total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 30250
```

Устранение пропусков в данных

```
hcols_with_na = [c for c in data.columns if data[c].isnull().sum() > 0]
# Количество пропусков
[(c, data[c].isnull().sum()) for c in hcols_with_na]
```

```
[('Name', 94),
 ('RawgID', 94),
 ('SteamURL', 55),
 ('Metacritic', 26894),
 ('Genres', 2968),
 ('Indie', 205),
 ('Presence', 94),
 ('Platform', 127),
 ('Graphics', 4320),
 ('Storage', 2759),
 ('Memory', 1934),
 ('RatingsBreakdown', 15206),
 ('ReleaseDate', 3226),
 ('Soundtrack', 205),
 ('Franchise', 25163),
 ('OriginalCost', 746),
 ('DiscountedCost', 29523),
 ('Players', 17916),
 ('Controller', 274),
 ('Languages', 223),
 ('ESRB', 25503),
 ('Achievements', 94),
 ('Publisher', 30250),
```

```
 ('Description', 219),
 ('Tags', 205)]
```

```
# Доля (процент) пропусков
[(c, data[c].isnull().mean()) for c in hcols_with_na]
```

```
[('Name', 0.0031074380165289255),
 ('RawgID', 0.0031074380165289255),
 ('SteamURL', 0.0018181818181818182),
 ('Metacritic', 0.8890578512396694),
 ('Genres', 0.09811570247933885),
 ('Indie', 0.006776859504132231),
 ('Presence', 0.0031074380165289255),
 ('Platform', 0.004198347107438017),
 ('Graphics', 0.1428099173553719),
 ('Storage', 0.09120661157024794),
 ('Memory', 0.06393388429752066),
 ('RatingsBreakdown', 0.5026776859504132),
 ('ReleaseDate', 0.10664462809917355),
 ('Soundtrack', 0.006776859504132231),
 ('Franchise', 0.8318347107438017),
 ('OriginalCost', 0.02466115702479339),
 ('DiscountedCost', 0.9759669421487603),
 ('Players', 0.5922644628099174),
 ('Controller', 0.009057851239669422),
 ('Languages', 0.007371900826446281),
 ('ESRB', 0.8430743801652892),
 ('Achievements', 0.0031074380165289255),
 ('Publisher', 1.0),
 ('Description', 0.007239669421487603),
 ('Tags', 0.006776859504132231)]
```

```
# Удаление колонки Publisher и Unnamed: 0 из-за неиспользования  в данной работе связей с другими датасетами
data = data.drop('Publisher', 1)
data = data.drop('Unnamed: 0', 1)
data.shape
```

```
<ipython-input-46-db29e5716c4b>:2: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argu
  data = data.drop('Publisher', 1)
<ipython-input-46-db29e5716c4b>:3: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argu
  data = data.drop('Unnamed: 0', 1)
(30250, 25)
```

```
# Колонки для которых удаляются пропуски
data = data.dropna(axis=0, subset=['Name', 'SteamURL'])
data.shape
```

```
(30101, 25)
```

```
hcols_with_na = [c for c in data.columns if data[c].isnull().sum() > 0]
# Количество пропусков
[(c, data[c].isnull().sum()) for c in hcols_with_na]
```

```
[('Metacritic', 26746),
 ('Genres', 2907),
 ('Indie', 176),
 ('Platform', 33),
 ('Graphics', 4250),
 ('Storage', 2697),
 ('Memory', 1872),
 ('RatingsBreakdown', 15112),
 ('ReleaseDate', 3132),
 ('Soundtrack', 176),
 ('Franchise', 25024),
 ('OriginalCost', 688),
 ('DiscountedCost', 29374),
 ('Players', 17813),
 ('Controller', 219),
 ('Languages', 168),
 ('ESRB', 25355),
 ('Description', 125),
 ('Tags', 176)]
```

Заполнение значений для одного признака

"Внедрение значений" - импьютация (imputation)

Обработка пропусков в числовых данных

```
# Выберем числовые колонки с пропущенными значениями
# Цикл по колонкам датасета
num_cols = []
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='float64' or dt=='int64'):
        num_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp_null_count, temp_perc))
```

```
Колонка Metacritic. Тип данных float64. Количество пустых значений 26746, 88.42%.
Колонка Indie. Тип данных float64. Количество пустых значений 176, 0.58%.
Колонка Soundtrack. Тип данных float64. Количество пустых значений 176, 0.58%.
Колонка Controller. Тип данных float64. Количество пустых значений 219, 0.72%.
```

```
# Фильтр по колонкам с пропущенными значениями
data_num = data[num_cols]
data_num
```

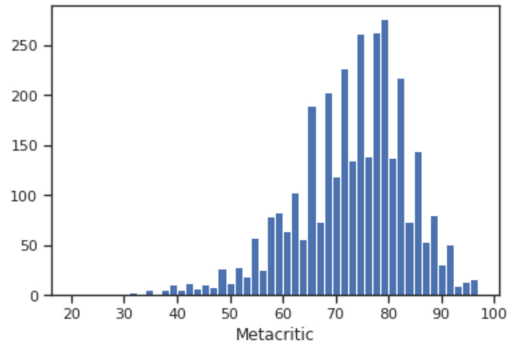
	Metacritic	Indie	Soundtrack	Controller
0	83.0	0.0	0.0	1.0
1	82.0	0.0	0.0	1.0
2	90.0	0.0	0.0	1.0
3	71.0	0.0	0.0	1.0
4	68.0	0.0	0.0	1.0
...	...	...	...	...
30245	NaN	1.0	0.0	1.0
30246	NaN	1.0	0.0	0.0
30247	NaN	0.0	0.0	0.0
30248	NaN	1.0	0.0	1.0
30249	NaN	0.0	0.0	1.0

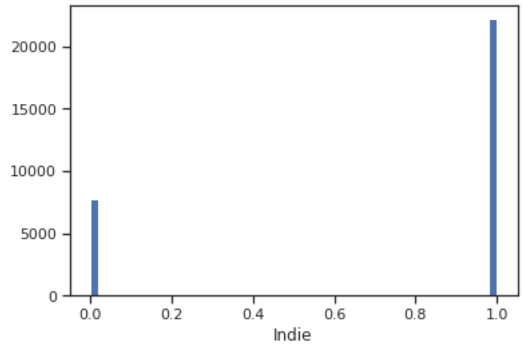
30101 rows × 4 columns

```
# Определим уникальные значения для полей
(data['Soundtrack'].unique(),
 data['Controller'].unique(),
 data['Indie'].unique())
```

```
(array([ 0.,  1., nan]), array([ 1.,  0., nan]), array([ 0.,  1., nan]))
```

```
# Гистограмма по признакам
for col in data_num:
    plt.hist(data[col], 50)
    plt.xlabel(col)
    plt.show()
```



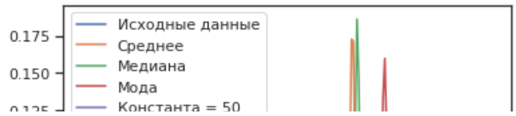



```
data_num_Metacritic = data_num[['Metacritic']]
data_num_Metacritic.head()
```

	Metacritic
0	83.0
1	82.0
2	90.0
3	71.0
4	68.0


```
def research_impute_numeric_column(dataset, num_column, const_value=None):
    strategy_params = ['mean', 'median', 'most_frequent', 'constant']
    strategy_params_names = ['Среднее', 'Медиана', 'Мода']
    strategy_params_names.append('Константа = ' + str(const_value))

    original_temp_data = dataset[[num_column]].values
    size = original_temp_data.shape[0]
    original_data = original_temp_data.reshape((size,))

    new_df = pd.DataFrame({'Исходные данные':original_data})

    for i in range(len(strategy_params)):
        strategy = strategy_params[i]
        col_name = strategy_params_names[i]
        if (strategy!='constant') or (strategy == 'constant' and const_value!=None):
            if strategy == 'constant':
                temp_data, _, _ = impute_column(dataset, num_column, strategy, fill_value_param=const_value)
            else:
                temp_data, _, _ = impute_column(dataset, num_column, strategy)
            new_df[col_name] = temp_data

    sns.kdeplot(data=new_df)
```

```
research_impute_numeric_column(data, 'Metacritic', 50)
```




```
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```

Попробуем заполнить пропущенные значения в колонке Metacritics значениями, вычисленными по среднему арифметическому, медиане и моде.

```
strategies=['mean', 'median', 'most_frequent']


def test_num_impute(strategy_param):
    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(data_num_Metacritic)
    return data_num_imp[mask_missing_values_only]


strategies[0], test_num_impute(strategies[0])
```

```
('mean', array([72.92280179, 72.92280179, 72.92280179, ..., 72.92280179,
        72.92280179, 72.92280179]))
```

```
strategies[1], test_num_impute(strategies[1])
```

```
('median', array([74., 74., 74., ..., 74., 74., 74.]))
```

```
strategies[2], test_num_impute(strategies[2])
```

```
('most_frequent', array([80., 80., 80., ..., 80., 80., 80.]))
```

```
# Более сложная функция, которая позволяет задавать колонку и вид импьютации
def test_num_impute_col(dataset, column, strategy_param):
    temp_data = dataset[[column]]

    indicator = MissingIndicator()
    mask_missing_values_only = indicator.fit_transform(temp_data)

    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(temp_data)

    filled_data = data_num_imp[mask_missing_values_only]

    return column, strategy_param, filled_data.size, filled_data[0], filled_data[filled_data.size-1]
```

```
data[['Metacritic']].describe()
```

	Metacritic
count	3355.000000
mean	72.922802
std	10.806216
min	20.000000
25%	67.000000
50%	74.000000
75%	80.000000
max	97.000000

```
test_num_impute_col(data, 'Metacritic', strategies[0])
```

```
('Metacritic', 'mean', 26746, 72.92280178837557, 72.92280178837557)
```

```
test_num_impute_col(data, 'Metacritic', strategies[1])
```

```
('Metacritic', 'median', 26746, 74.0, 74.0)
```

```
test_num_impute_col(data, 'Metacritic', strategies[2])
```

```
('Metacritic', 'most_frequent', 26746, 80.0, 80.0)
```

Обработка пропусков в категориальных данных

```
# Выберем категориальные колонки с пропущенными значениями
# Цикл по колонкам датасета
cat_cols = []
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='object'):
        cat_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp_null_count, temp_perc))
```

```
Колонка Genres. Тип данных object. Количество пустых значений 2907, 9.61%.
Колонка Platform. Тип данных object. Количество пустых значений 33, 0.11%.
Колонка Graphics. Тип данных object. Количество пустых значений 4250, 14.05%.
Колонка Storage. Тип данных object. Количество пустых значений 2697, 8.92%.
Колонка Memory. Тип данных object. Количество пустых значений 1872, 6.19%.
Колонка RatingsBreakdown. Тип данных object. Количество пустых значений 15112, 49.96%.
Колонка ReleaseDate. Тип данных object. Количество пустых значений 3132, 10.35%.
Колонка Franchise. Тип данных object. Количество пустых значений 25024, 82.72%.
Колонка OriginalCost. Тип данных object. Количество пустых значений 688, 2.27%.
Колонка DiscountedCost. Тип данных object. Количество пустых значений 29374, 97.1%.
Колонка Players. Тип данных object. Количество пустых значений 17813, 58.89%.
Колонка Languages. Тип данных object. Количество пустых значений 168, 0.56%.
Колонка ESRB. Тип данных object. Количество пустых значений 25355, 83.82%.
Колонка Description. Тип данных object. Количество пустых значений 125, 0.41%.
Колонка Tags. Тип данных object. Количество пустых значений 176, 0.58%.
```

- Колонки, содержащие менее 5% пропусков выбираем для построения модели.
- Колонки, содержащие менее 30% пропусков также выбираем для построения модели.
- Колонки RatingsBreakdown (49.96%) и Players (59.07%) не выбираем для построения модели, в случае отсутствия необходимости в этих колонках.
- Колонки Franchise (82.91%), DiscountedCost (97.29%) и ESRB (84.0%) не выбираем для построения модели в любом случае.

```
cat_temp_data = data[['Genres']]
cat_temp_data.head()
```

	Genres
0	Action, Free to Play
1	Action, Adventure, Free to Play
2	NaN
3	Massively Multiplayer, RPG
4	Action, Adventure

```
cat_temp_data['Genres'].unique()
```

```
array(['Action, Free to Play', 'Action, Adventure, Free to Play', nan,
       ..., 'Casual, Indie, Massively Multiplayer, RPG, Early Access',
       'Action, Adventure, Casual, Racing, Simulation, Strategy',
       'Action, Adventure, Casual, Sports, Strategy'], dtype=object)
```

```
cat_temp_data[cat_temp_data['Genres'].isnull()].shape
```

```
(2907, 1)
```

```
# Импьютация наиболее частыми значениями
imp2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data_imp2 = imp2.fit_transform(cat_temp_data)
data_imp2
```

```
array([['Action, Free to Play'],
       ['Action, Adventure, Free to Play'],
       ['Action, Indie'],
       ...,
       ['Casual'],
       ['Action, Adventure, Casual, Indie'],
       ['Action, Indie']], dtype=object)
```

```
# Пустые значения отсутствуют
np.unique(data_imp2)
```

```
array(['Action', 'Action, Adventure', 'Action, Adventure, Casual', ...,
       'Strategy, Indie, Casual, Simulation', 'Strategy, RPG, Indie',
       'Strategy, Simulation'], dtype=object)
```

```
# Импьютация константой
imp3 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value='NA')
data_imp3 = imp3.fit_transform(cat_temp_data)
data_imp3
```

```
array([['Action, Free to Play'],
       ['Action, Adventure, Free to Play'],
       ['NA'],
       ...,
       ['Casual'],
       ['Action, Adventure, Casual, Indie'],
       ['NA']], dtype=object)
```

```
np.unique(data_imp3)
```

```
array(['Action', 'Action, Adventure', 'Action, Adventure, Casual', ...,
       'Strategy, Indie, Casual, Simulation', 'Strategy, RPG, Indie',
       'Strategy, Simulation'], dtype=object)
```

```
data_imp3[data_imp3=='NA'].size
```

```
2907
```

Таким образом, в колонку Genres вставлено 2962 "NA", вместо пропущенных значений.

Преобразование категориальных признаков в числовые

```
cat_enc = pd.DataFrame({'c1':data_imp2.T[0]})
cat_enc
```

	c1
0	Action, Free to Play
1	Action, Adventure, Free to Play
2	Action, Indie
3	Massively Multiplayer, RPG
4	Action, Adventure
...	...
30096	Casual, Indie
30097	Indie
30098	Casual
30099	Action, Adventure, Casual, Indie
30100	Action, Indie

30101 rows × 1 columns

Кодирование категорий целочисленными значениями - label encoding

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

```
le = LabelEncoder()
cat_enc_le = le.fit_transform(cat_enc['c1'])
```

```
cat_enc['c1'].unique()
```

```
array(['Action, Free to Play', 'Action, Adventure, Free to Play',
       'Action, Indie', ...,
       'Casual, Indie, Massively Multiplayer, RPG, Early Access',
       'Action, Adventure, Casual, Racing, Simulation, Strategy',
       'Action, Adventure, Casual, Sports, Strategy'], dtype=object)
```

```
np.unique(cat_enc_le)
```

```
array([   0,    1,    2, ..., 1003, 1004, 1005])
```

Кодирование категорий наборами бинарных значений - one-hot encoding

```
ohe = OneHotEncoder()
cat_enc_ohe = ohe.fit_transform(cat_enc[['c1']])
```

```
cat_enc.shape
```

```
(30101, 1)
```

```
cat_enc_ohe.shape
```

```
(30101, 1006)
```

```
cat_enc_ohe
```

```
<30101x1006 sparse matrix of type '<class 'numpy.float64'>'
	with 30101 stored elements in Compressed Sparse Row format>
```

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

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

```
cat_enc.head(10)
```

	c1
0	Action, Free to Play
1	Action, Adventure, Free to Play
2	Action, Indie
3	Massively Multiplayer, RPG
4	Action, Adventure
5	Adventure, Indie, Simulation, Strategy, Early ...
6	Action
7	Action, Indie, Racing, Sports
8	Action
9	Action, Adventure, Indie, Massively Multiplaye...

Pandas get_dummies - быстрый вариант one-hot кодирования

```
pd.get_dummies(cat_enc).head()
```

c1_Action,	c1_Action, Adventure,	c1_Action, Adventure, Casual,	c1_Action, Adventure, Casual,	c1_Action, Adventure, Casual, Free	c1_Action, Adventure, Casual, Free	c1_Action, Adventure, Casual, Free

```
pd.get_dummies(cat_temp_data, dummy_na=True).head()
```

	Genres_Action	Genres_Action, Adventure	Genres_Action, Adventure, Casual	Genres_Action, Adventure, Casual, Early Access	Genres_Action, Adventure, Casual, Free to Play, Indie	Genres_Action, Adventure, Casual, Free to Play, Indie, Early Access	Genres_Action, Adventure, Casual, Free to Play, Indie, Massively Multiplayer	Genres_Action, Adventure, Casual, Free to Play, Indie, Massively Multiplayer, RPG	Ge
0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	
4	0	1	0	0	0	0	0	0	

5 rows × 1007 columns

Count (frequency) encoding

```
!pip install category_encoders
from category_encoders.count import CountEncoder as ce_CountEncoder
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting category_encoders
  Downloading category_encoders-2.6.0-py2.py3-none-any.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.2/81.2 KB 7.5 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.9/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.9/dist-packages (from category_encoders) (0.5.3)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.9/dist-packages (from category_encoders) (0.13.5)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.9/dist-packages (from category_encoders) (1.4.4)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from category_encoders) (1.10.1)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.9/dist-packages (from category_encoders) (1.22.4)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.5->category_encod
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=1.0.5->category_encoders) (2022
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn>=0.20.0->category_
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-packages (from scikit-learn>=0.20.0->category_encoder
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.9/dist-packages (from statsmodels>=0.9.0->category_encoder
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.0
```

```
ce_CountEncoder1 = ce_CountEncoder()
data_COUNT_ENC = ce_CountEncoder1.fit_transform(data[data.columns.difference(['Genres'])])
data_COUNT_ENC
```

	Achievements	Controller	Description	DiscountedCost	ESRB	Franchise	Graphics	Indie	Languages	Memory	...	Players	Pr
0	179.0	1.0	1	29374	776	25024	8	0.0	1	6462	...	65	10

```
ce_CountEncoder2 = ce_CountEncoder(normalize=True)
data_FREQ_ENC = ce_CountEncoder2.fit_transform(data[data.columns.difference(['Genres'])])
data_FREQ_ENC
```

	Achievements	Controller	Description	DiscountedCost	ESRB	Franchise	Graphics	Indie	Languages	Memory	...	Players
0	179.0	1.0	0.000033	0.975848	0.025780	0.831335	0.000266	0.0	0.000033	0.214677	...	0.00215
1	61.0	1.0	0.000033	0.975848	0.047374	0.000033	0.000066	0.0	0.004219	0.015614	...	0.00461
2	0.0	1.0	0.000033	0.975848	0.842331	0.831335	0.000764	0.0	0.000033	0.207568	...	0.00215
3	0.0	1.0	0.000033	0.975848	0.842331	0.000066	0.000100	0.0	0.010697	0.011960	...	0.01784
4	308.0	1.0	0.000033	0.975848	0.047374	0.831335	0.000033	0.0	0.008538	0.207568	...	0.00215
...	...	...	...	...	...	...	...	...	...	...	...	..
30245	0.0	1.0	0.000033	0.975848	0.842331	0.831335	0.000664	1.0	0.567423	0.094249	...	0.59177
30246	0.0	0.0	0.000033	0.975848	0.842331	0.831335	0.000233	1.0	0.567423	0.015614	...	0.59177
30247	0.0	0.0	0.000033	0.975848	0.842331	0.000033	0.141191	0.0	0.567423	0.009667	...	0.59177
30248	0.0	1.0	0.000033	0.975848	0.842331	0.831335	0.141191	1.0	0.567423	0.008172	...	0.59177
30249	0.0	1.0	0.000033	0.975848	0.842331	0.831335	0.000897	0.0	0.567423	0.207568	...	0.59177

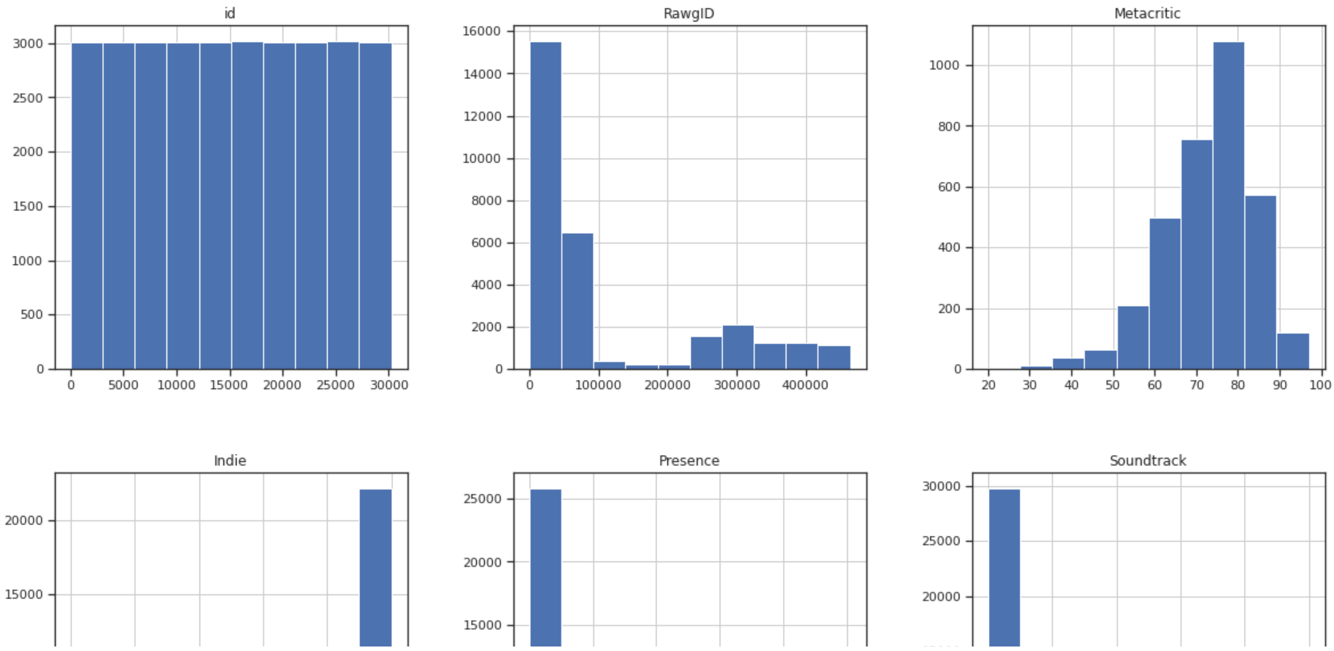
30101 rows × 24 columns

Нормализация числовых признаков

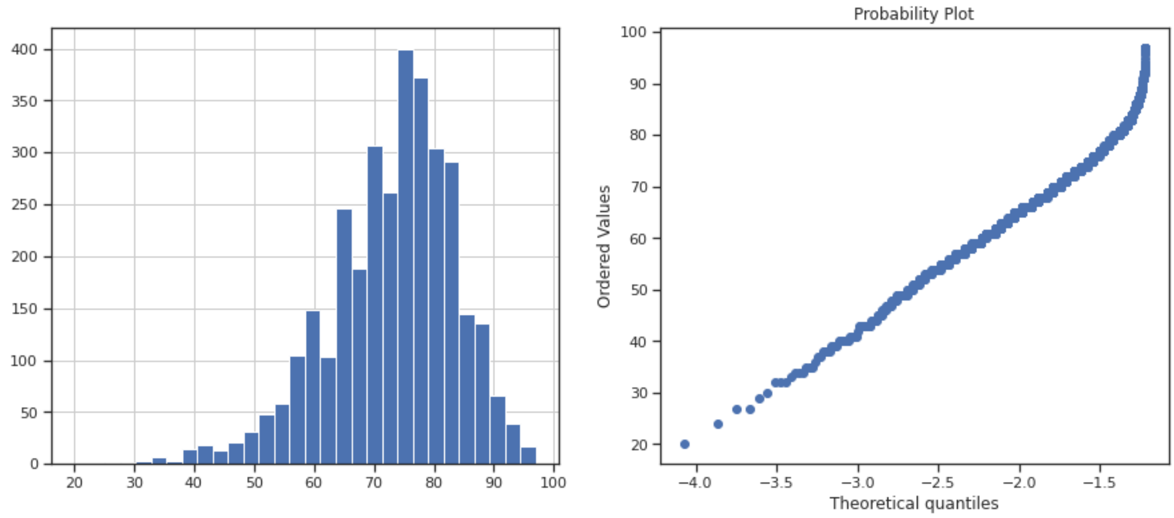
```
import scipy.stats as stats
```

```
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()
```

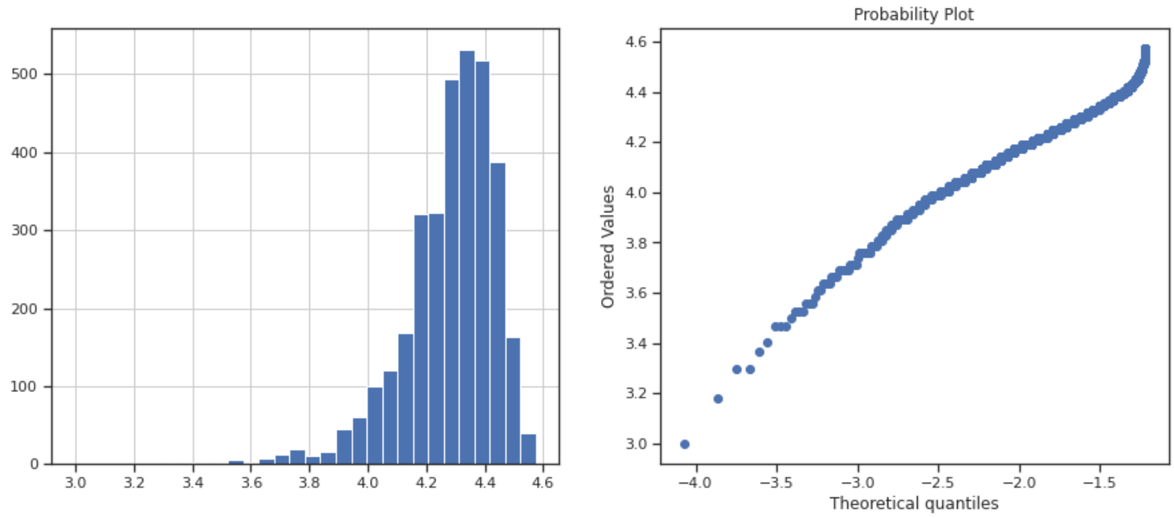
```
data.hist(figsize=(20,20))
plt.show()
```



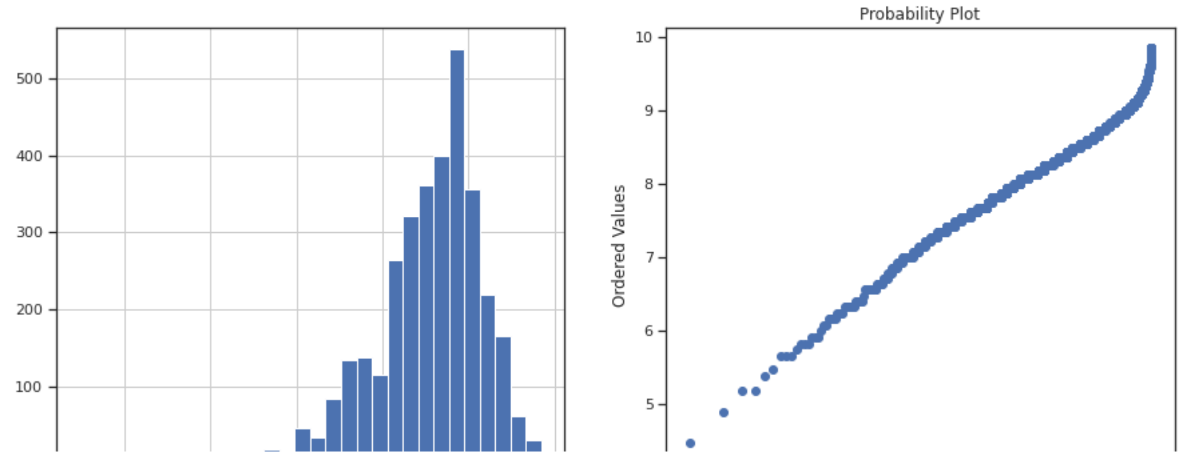

```
diagnostic_plots(data, 'Metacritic')
```



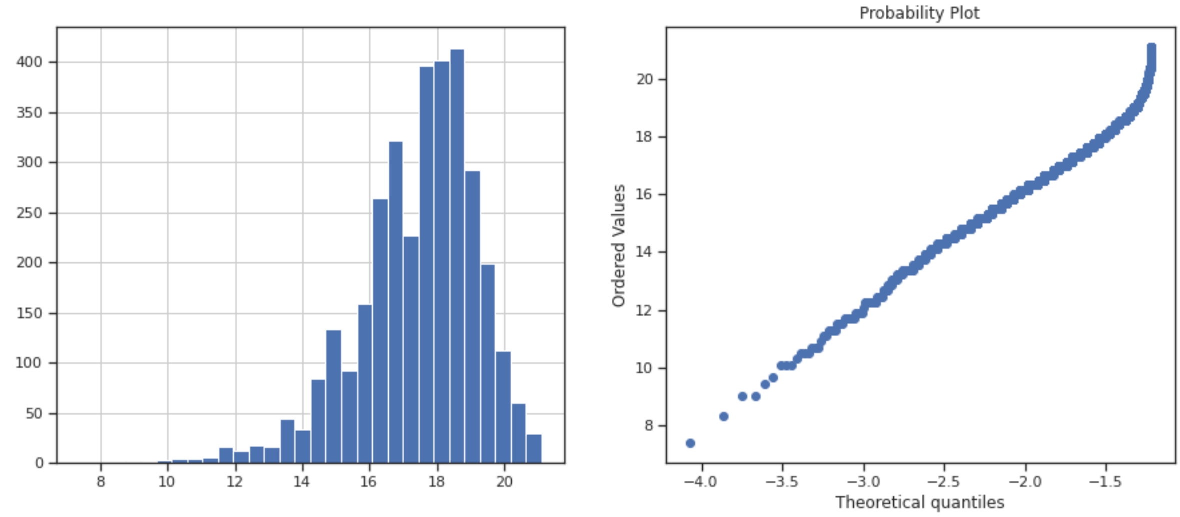

```
data['Metacritic_log'] = np.log(data['Metacritic'])
diagnostic_plots(data, 'Metacritic_log')
```



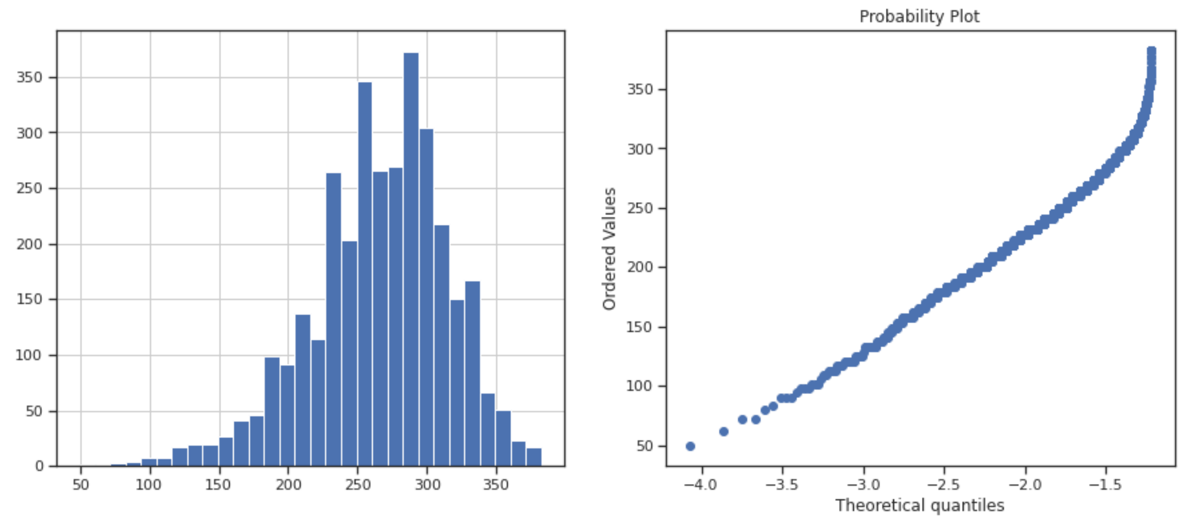

```
data['Metacritic_sqr'] = data['Metacritic']**(1/2)
diagnostic_plots(data, 'Metacritic_sqr')
```




```
data['Metacritic_exp1'] = data['Metacritic']**(1/1.5)
diagnostic_plots(data, 'Metacritic_exp1')
```



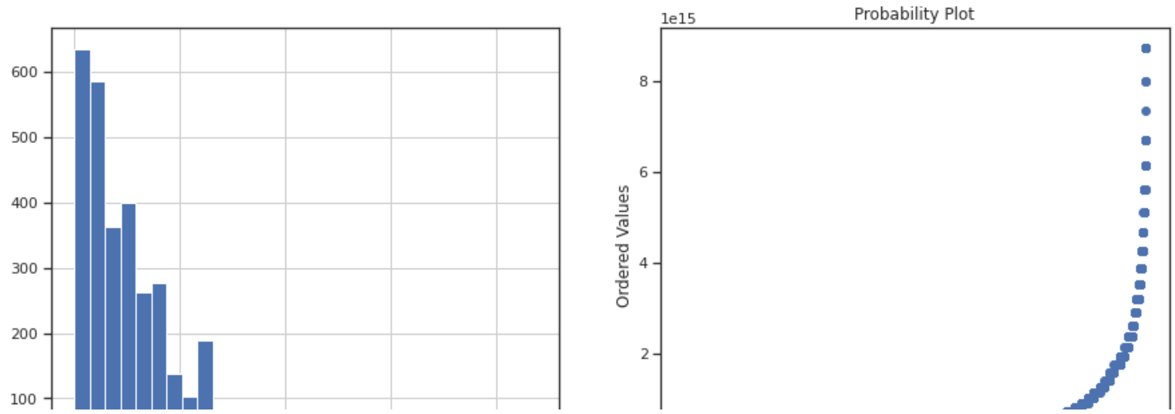

```
data['Metacritic_exp2'] = data['Metacritic']**(1.3)
diagnostic_plots(data, 'Metacritic_exp2')
```



Не очень хорошие результаты:

```
data['Metacritic'] = data['Metacritic'].astype('float')
data['Metacritic_yeojohnson'], param = stats.yeojohnson(data['Metacritic'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(data, 'Metacritic_yeojohnson')
```

Оптимальное значение λ = 8.472135811722177




```
data['Metacritic_boxcox'], param = stats.boxcox(data['Metacritic'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(data, 'Metacritic_boxcox')
```

Оптимальное значение λ = 8.472135811722177