Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления и искусственный интеллект

КАФЕДРА Системы обработки информации и управления

Лабораторная работа №2
По курсу
«Методы машинного обучения в АСОИУ»

«Обработка признаков, часть 1»

Выполнил:

ИУ5-22М Киричков Е. Е.

20.05.2024

Проверил:

Балашов А.М.

Москва, 2024

Задание:

1. 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.
2. Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи:
   - устранение пропусков в данных;
   - кодирование категориальных признаков;
   - нормализация числовых признаков.

Ввод [1]:
```
# Импортирование библиотек
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
import plotly.express as px
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
%matplotlib inline
sns.set(style="ticks")
```

Ввод [2]:
```
data_loaded = pd.read_csv("data/Gender_Inequality_Index.csv")
```

Ввод [3]:
```
data = data_loaded
```

i. Устранение пропусков в данных

Ввод [4]:
```
# Названия стобцов и их тип данных, у которых есть пропуски
cols_with_na = [(i, data[i].dtypes) for i in data.columns if data[i].isna()
cols_with_na
```

Out[4]:
```
[('Human_development', dtype('O')),
 ('GII', dtype('float64')),
 ('Rank', dtype('float64')),
 ('Maternal_mortality', dtype('float64')),
 ('Seats_parliament', dtype('float64')),
 ('F_secondary_educ', dtype('float64')),
 ('M_secondary_educ', dtype('float64')),
 ('F_Labour_force', dtype('float64')),
 ('M_Labour_force', dtype('float64'))]
```

Ввод [5]:
```
# есть ли отрицательные значения?
[i for i in data.select_dtypes(include=['int', 'float64']).columns if (data
```

Out[5]:
```
[]
```

Ввод [6]:
```
cols_with_all = [(i, data[i].dtypes) for i in data.columns]
cols_with_all
```

Out[6]:
```
[('Country', dtype('O')),
 ('Human_development', dtype('O')),
 ('GII', dtype('float64')),
 ('Rank', dtype('float64')),
 ('Maternal_mortality', dtype('float64')),
 ('Adolescent_birth_rate', dtype('float64')),
 ('Seats_parliament', dtype('float64')),
 ('F_secondary_educ', dtype('float64')),
 ('M_secondary_educ', dtype('float64')),
 ('F_Labour_force', dtype('float64')),
 ('M_Labour_force', dtype('float64'))]
```

Ввод [7]:
```
# Вывод процент пропусков
[(c, round(data[c].isnull().mean() * 100, 4)) for c in data.columns if data
```

Out[7]:
```
[('Human_development', 2.0513),
 ('GII', 12.8205),
 ('Rank', 12.8205),
 ('Maternal_mortality', 5.641),
 ('Seats_parliament', 1.0256),
 ('F_secondary_educ', 9.2308),
 ('M_secondary_educ', 9.2308),
 ('F_Labour_force', 7.6923),
 ('M_Labour_force', 7.6923)]
```

Ввод [8]:
```
data.shape
```

Out[8]:
```
(195, 11)
```

Ввод [9]:
```
data.head()
```

Out[9]:

	Country	Human_development	GII	Rank	Maternal_mortality	Adolescent_birth_rate	Seats_parl
0	Switzerland	Very high	0.018	3.0	5.0	2.2	
1	Norway	Very high	0.016	2.0	2.0	2.3	
2	Iceland	Very high	0.043	8.0	4.0	5.4	
3	Hong Kong	Very high	NaN	NaN	NaN	1.6	
4	Australia	Very high	0.073	19.0	6.0	8.1	

Ввод [10]:
```
data.describe()
```

Out[10]:

	GII	Rank	Maternal_mortality	Adolescent_birth_rate	Seats_parliament	F_seconda
count	170.000000	170.000000	184.000000	195.000000	193.000000	177
mean	0.344376	85.376471	160.027174	44.597949	24.701554	62
std	0.197105	49.210206	233.028867	38.422479	12.404319	29
min	0.013000	1.000000	2.000000	1.600000	0.000000	6
25%	0.177500	43.250000	12.750000	10.750000	16.500000	37
50%	0.363000	85.500000	53.000000	36.200000	23.600000	69
75%	0.505750	127.750000	188.250000	64.200000	33.600000	90
max	0.820000	170.000000	1150.000000	170.500000	55.700000	100

1. Удалять пропущенные значения нецелесообразно ввиду небольшого размера датасета, поэтому проведем замену пропусков различными методами.

Ввод [11]:
```
res = data.dropna(axis=1, how='all')
res.shape
# Размер не изменился, пустых строчек нет для удаления
```

Out[11]:
```
(195, 11)
```

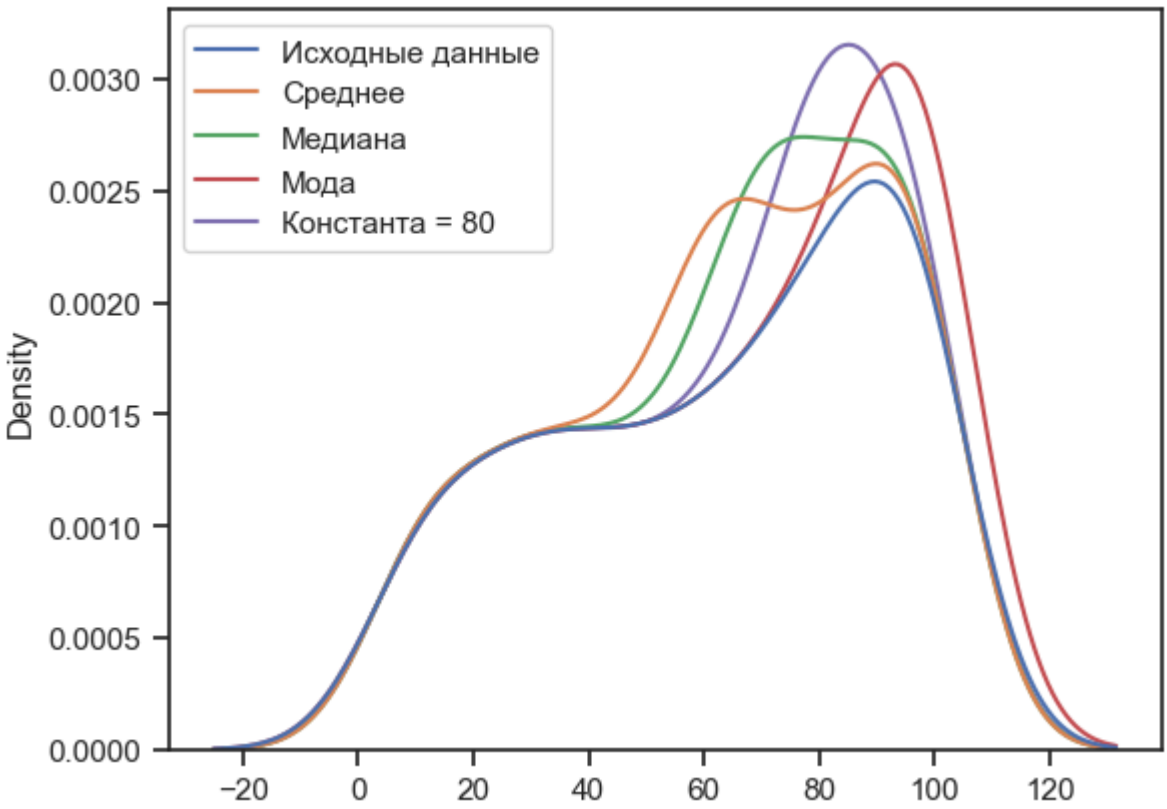
2. Заполнение показателями центра распределения и константой

Ввод [12]:
```
# Для внедрения значения используется класс SimpleImputer
def impute_column(dataset, column, strategy_param, fill_value_param=None):
    """
    Заполнение пропусков в одном признаке
    """
    temp_data = dataset[[column]].values
    size = temp_data.shape[0]
    imputer = SimpleImputer(strategy=strategy_param,
                            fill_value=fill_value_param)
    all_data = imputer.fit_transform(temp_data)
    return all_data.reshape((size,))
```

Ввод [13]:
```
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_valu
            if strategy == 'constant':
                temp_data = impute_column(dataset, num_column, strategy, fi
            else:
                temp_data = impute_column(dataset, num_column, strategy)
            new_df[col_name] = temp_data

    sns.kdeplot(data=new_df)
```

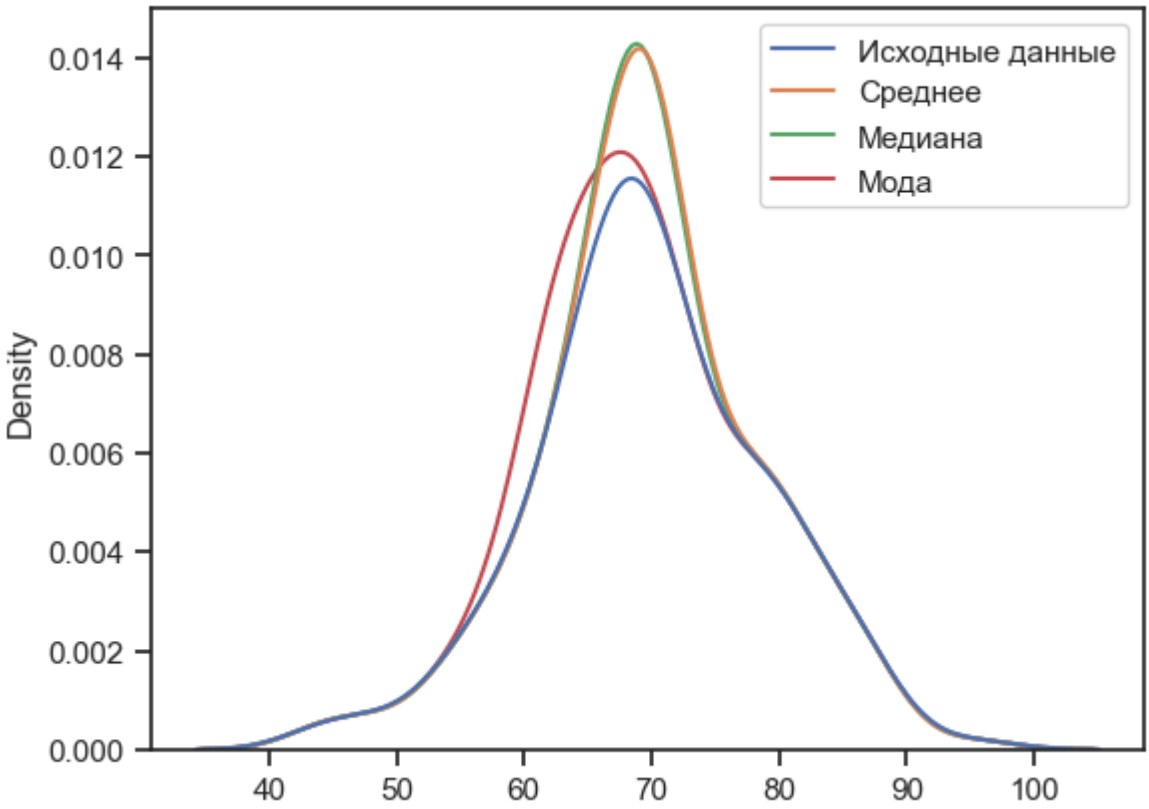
Ввод [48]:
```
research_impute_numeric_column(data, 'F_secondary_educ', 80)
```

```
/Users/evseykirichkov/anaconda3/lib/python3.11/site-packages/seaborn/_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
removed in a future version. Convert inf values to NaN before operating in
stead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Ввод [15]:
```
research_impute_numeric_column(data, 'M_Labour_force')
```

```
/Users/evseykirichkov/anaconda3/lib/python3.11/site-packages/seaborn/_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
removed in a future version. Convert inf values to NaN before operating in
stead.
  with pd.option_context('mode.use_inf_as_na', True):
```



3. Заполнение "хвостом распределения"

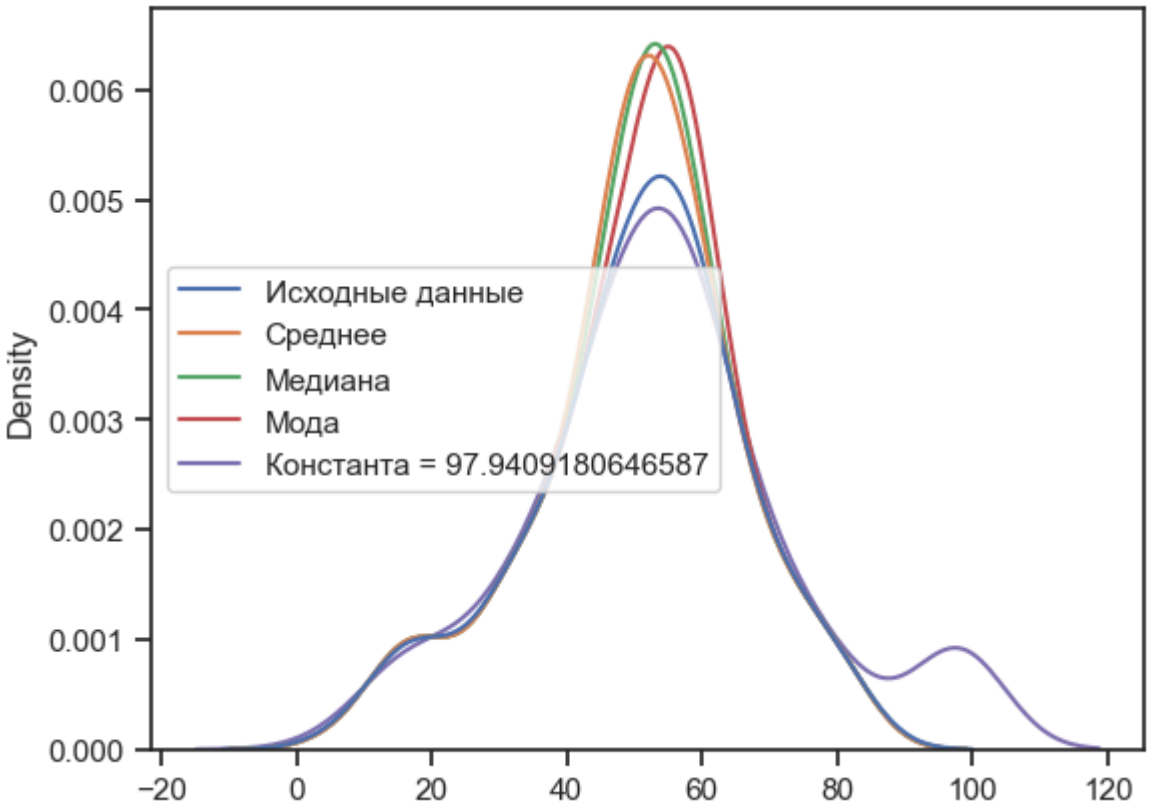
Параметр F_Labour_force, обозначающий уровень участия женщин в рабочей силе, больше похож на нормальной распределение. Определим для него верхнюю границу экстремальных значений в данных (extreme_value).

Ввод [16]:
```
F_Labour_force_ev = data['F_Labour_force'].mean() + 3*data['F_Labour_force'
F_Labour_force_ev
```

Out[16]:
```
97.9409180646587
```

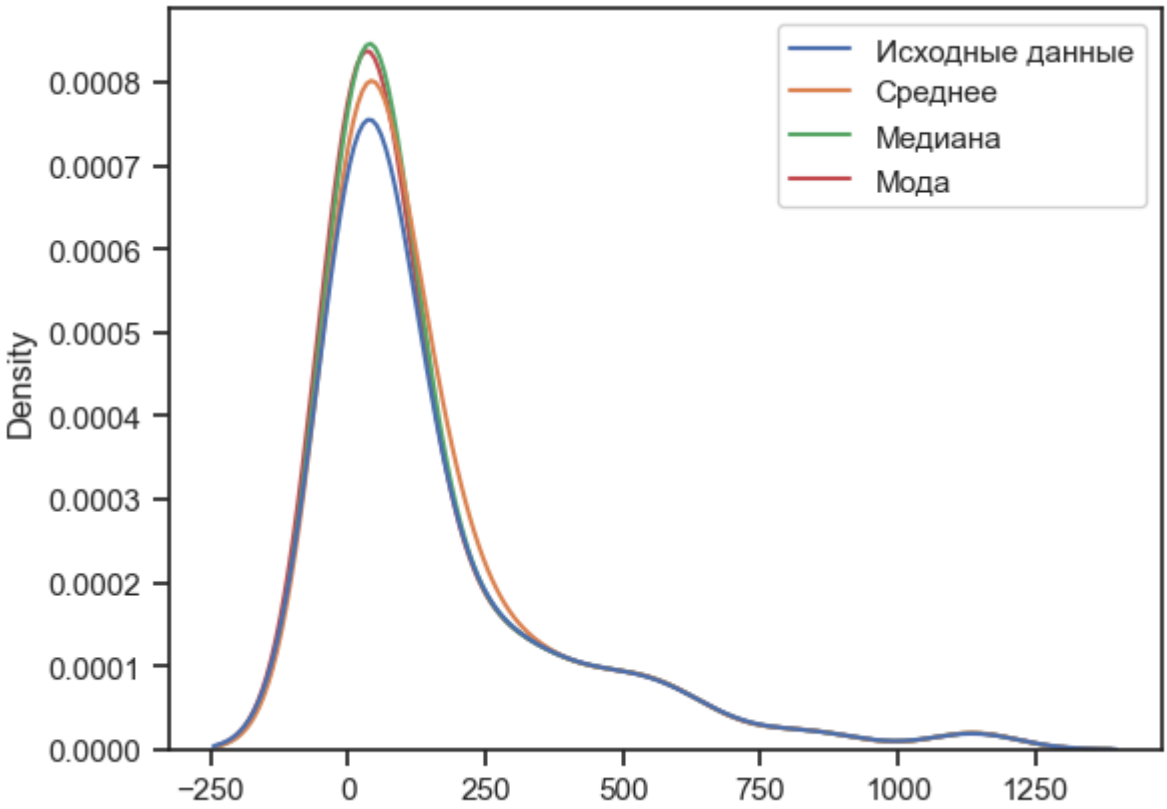
Ввод [17]:
```
research_impute_numeric_column(data, 'F_Labour_force', F_Labour_force_ev)
```

```
/Users/evseykirichkov/anaconda3/lib/python3.11/site-packages/seaborn/_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
removed in a future version. Convert inf values to NaN before operating in
stead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Ввод [18]:
```
research_impute_numeric_column(data, 'Maternal_mortality')
```

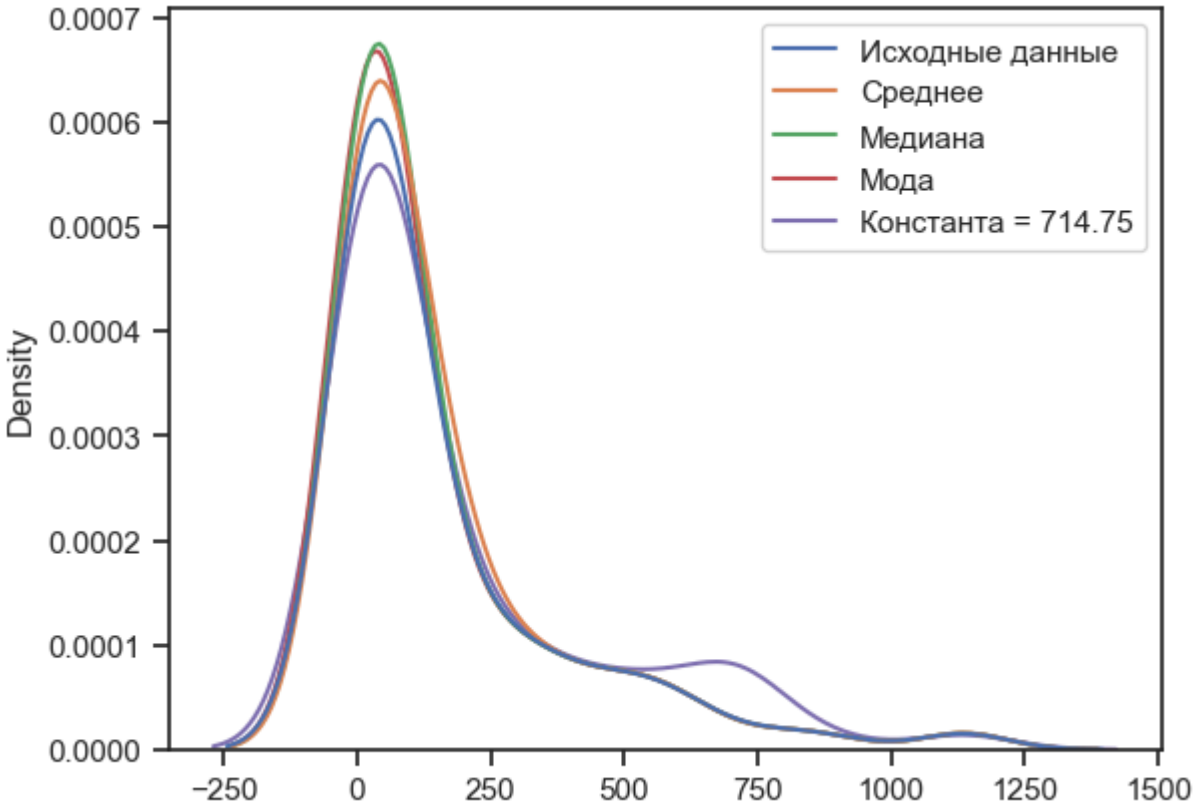
```
/Users/evseykirichkov/anaconda3/lib/python3.11/site-packages/seaborn/_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
removed in a future version. Convert inf values to NaN before operating in
stead.
  with pd.option_context('mode.use_inf_as_na', True):
```



На ассиметричное распределение больше похоже распределение параметра Maternal_mortality, обозначающий материнскую смертность.

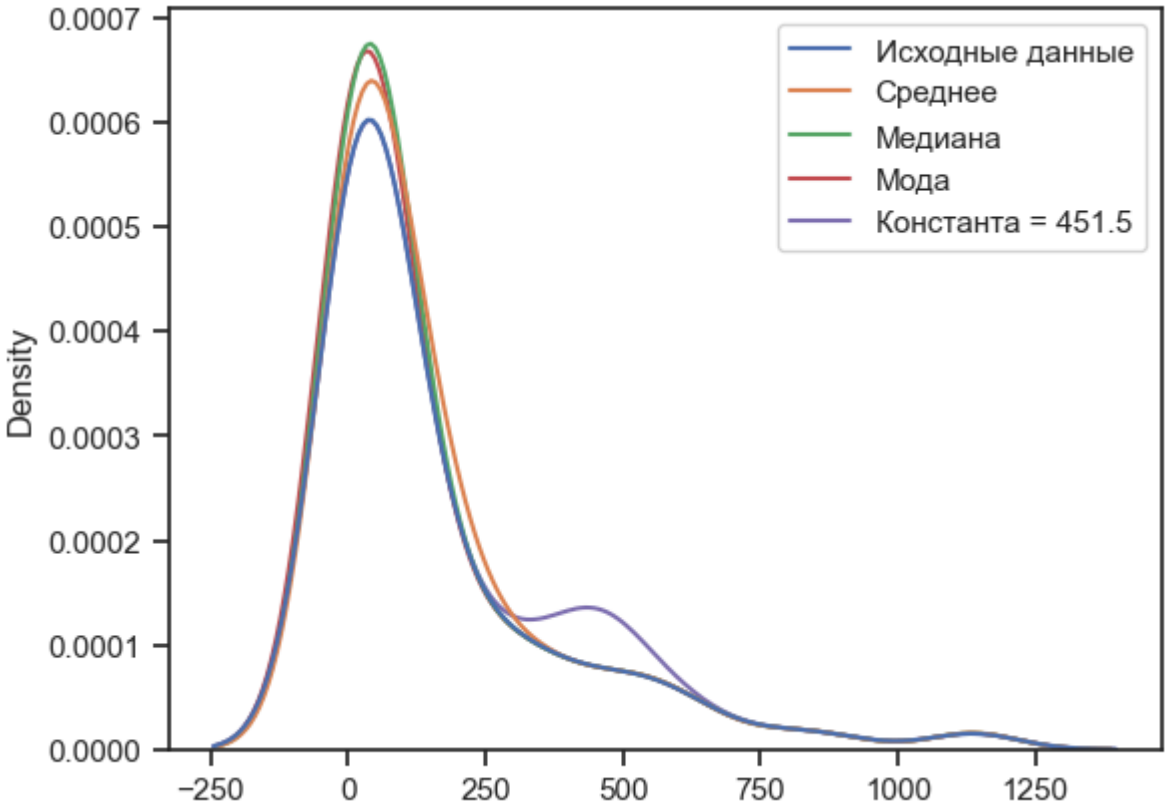
Ввод [19]:
```
IQR = data['Maternal_mortality'].quantile(0.75) - data['Maternal_mortality'
Maternal_mortality_ev1 = data['Maternal_mortality'].quantile(0.75) + 3*IQR
print(f'IQR={IQR}, extreme_value={Maternal_mortality_ev1}')
research_impute_numeric_column(data, 'Maternal_mortality', Maternal_mortali
```

```
IQR=175.5, extreme_value=714.75

/Users/evseykirichkov/anaconda3/lib/python3.11/site-packages/seaborn/_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
removed in a future version. Convert inf values to NaN before operating in
stead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Ввод [20]:
```
Maternal_mortality_ev2 = data['Maternal_mortality'].quantile(0.75) + 1.5*IQ
print(f'IQR={IQR}, extreme_value={Maternal_mortality_ev2}')
research_impute_numeric_column(data, 'Maternal_mortality', Maternal_mortali
```

```
IQR=175.5, extreme_value=451.5

/Users/evseykirichkov/anaconda3/lib/python3.11/site-packages/seaborn/_oldc
ore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be
removed in a future version. Convert inf values to NaN before operating in
stead.
  with pd.option_context('mode.use_inf_as_na', True):
```

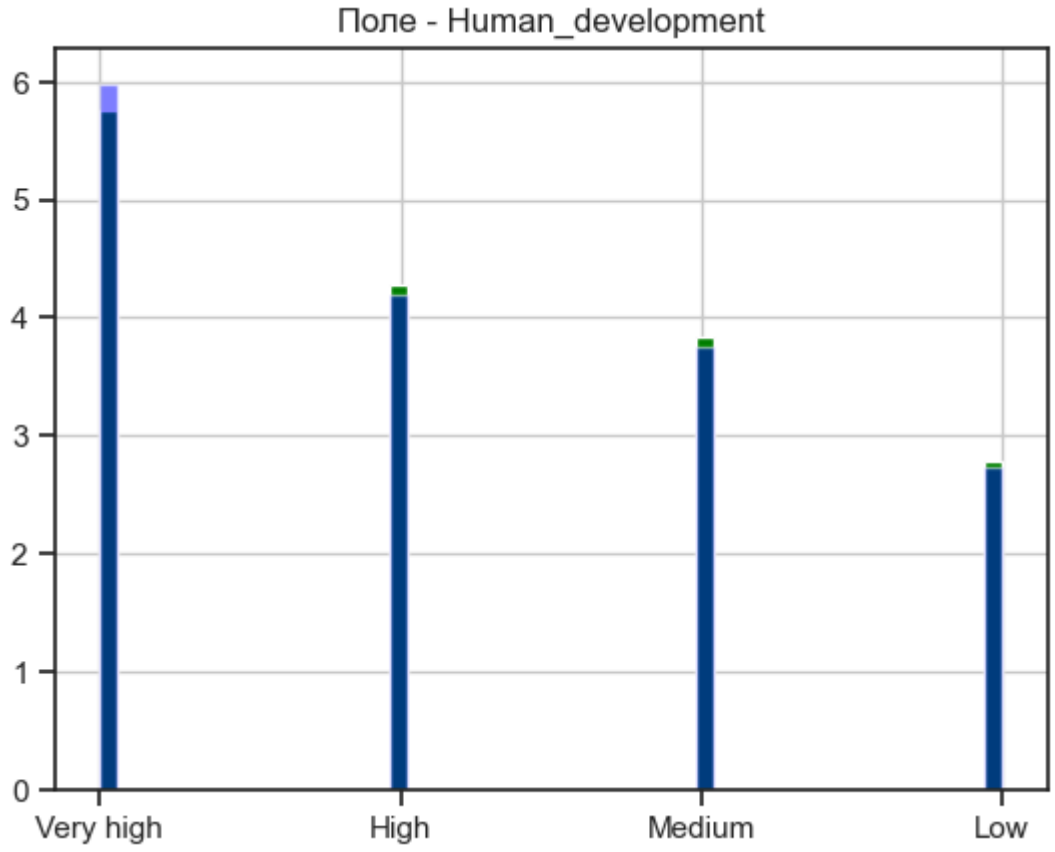


3. Заполнение наиболее распространенным значением категории

Единственный категориальный признак с пропусками в данном датасете - это Human_development, обозначающий категорию человеческого развития в стране: Низкий-Очень высокий.

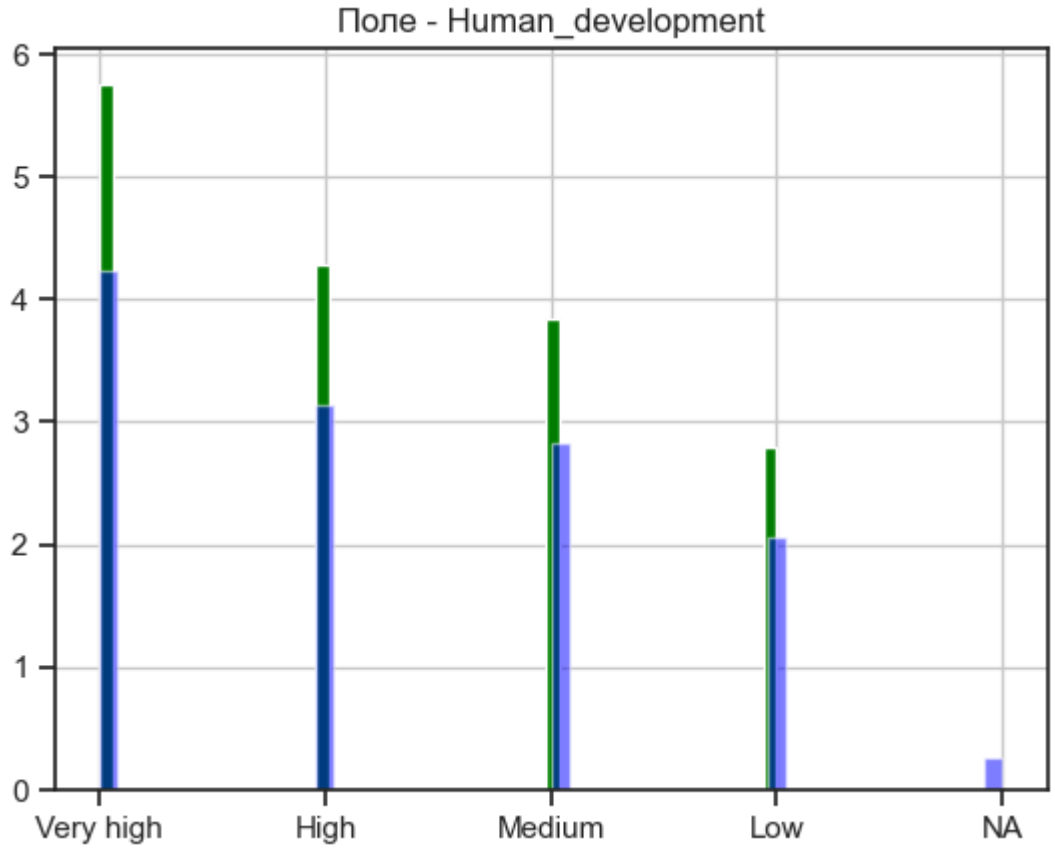
Ввод [21]:
```
def plot_hist_diff(old_ds, new_ds, col):
    """
    Разница между распределениями до и после устранения пропусков
    """
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.title.set_text('Поле - ' + str(col))
    old_ds[col].hist(bins=50, ax=ax, density=True, color='green')
    new_ds[col].hist(bins=50, ax=ax, color='blue', density=True, alpha=0.5)
    plt.show()
```

Ввод [22]:
```
data_cat_new = data[['Human_development']].copy()
Human_development_cat_new_temp = impute_column(data_cat_new, 'Human_develop
data_cat_new['Human_development'] = Human_development_cat_new_temp
plot_hist_diff(data, data_cat_new, 'Human_development')
```



4. Введение отдельного значения категории для пропущенных значений

Ввод [23]:
```
data_cat_na = data[['Human_development']].copy()
Human_development_cat_na = impute_column(data_cat_na, 'Human_development',
data_cat_na['Human_development'] = Human_development_cat_na
plot_hist_diff(data, data_cat_na, 'Human_development')
```



ii. Кодирование категориальных признаков

Ввод [24]:
```
data_loaded_2 = pd.read_csv("data/retail_sales_dataset 2.csv")
data_2 = data_loaded_2
```

Ввод [25]:
```
data_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Transaction ID    1000 non-null   int64
 1   Date              1000 non-null   object
 2   Customer ID       1000 non-null   object
 3   Gender            1000 non-null   object
 4   Age               1000 non-null   int64
 5   Product Category  1000 non-null   object
 6   Quantity          1000 non-null   int64
 7   Price per Unit    1000 non-null   int64
 8   Total Amount      1000 non-null   int64
dtypes: int64(5), object(4)
memory usage: 70.4+ KB
```

Ввод [26]:
```
data_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Transaction ID    1000 non-null   int64 
 1   Date              1000 non-null   object
 2   Customer ID       1000 non-null   object
 3   Gender            1000 non-null   object
 4   Age               1000 non-null   int64 
 5   Product Category  1000 non-null   object
 6   Quantity          1000 non-null   int64 
 7   Price per Unit    1000 non-null   int64 
 8   Total Amount      1000 non-null   int64 
dtypes: int64(5), object(4)
memory usage: 70.4+ KB
```

Ввод [27]:
```
# Названия стобцов и их тип данных, у которых есть пропуски
cols_2_with_na = [(i, data_2[i].dtypes) for i in data_2.columns if data_2[i
cols_2_with_na
```

Out[27]:
```
[]
```

Пропуски не обнаружены.

Кодирование категорий целочисленными значениями - label encoding

Ввод [28]:
```
data_2['Product Category'].unique()
```

Out[28]:
```
array(['Beauty', 'Clothing', 'Electronics'], dtype=object)
```

Ввод [29]:
```
le = LabelEncoder()
cat_enc_le = le.fit_transform(data_2['Product Category'])
```

Ввод [30]:
```
np.unique(cat_enc_le)
```

Out[30]:
```
array([0, 1, 2])
```

Ввод [31]:
```
le.inverse_transform([0, 1])
```

Out[31]:
```
array(['Beauty', 'Clothing'], dtype=object)
```

Кодирование категорий наборами бинарных значений - one-hot encoding

Ввод [32]:
```
ohe = OneHotEncoder()
cat_enc_ohe = ohe.fit_transform(data_2[['Product Category']])
cat_enc_ohe
```

Out[32]:
```
<1000x3 sparse matrix of type '<class 'numpy.float64'>'
        with 1000 stored elements in Compressed Sparse Row format>
```

Ввод [33]:
```
cat_enc_ohe.todense()[0:10]
```

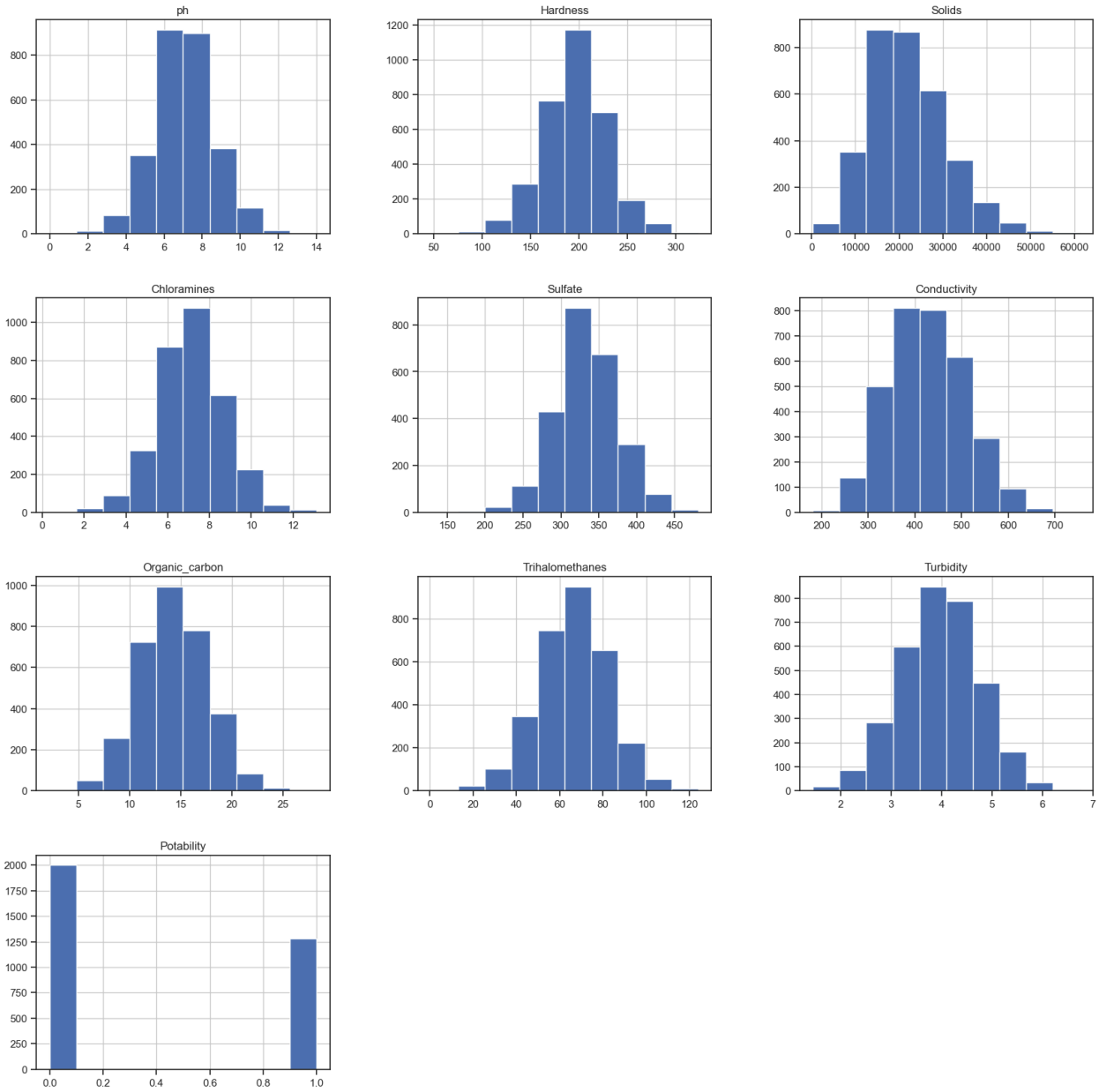
Out[33]:
```
matrix([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.],
        [0., 1., 0.],
        [1., 0., 0.],
        [1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.],
        [0., 0., 1.],
        [0., 1., 0.]])
```

iii. Нормализация числовых признаков

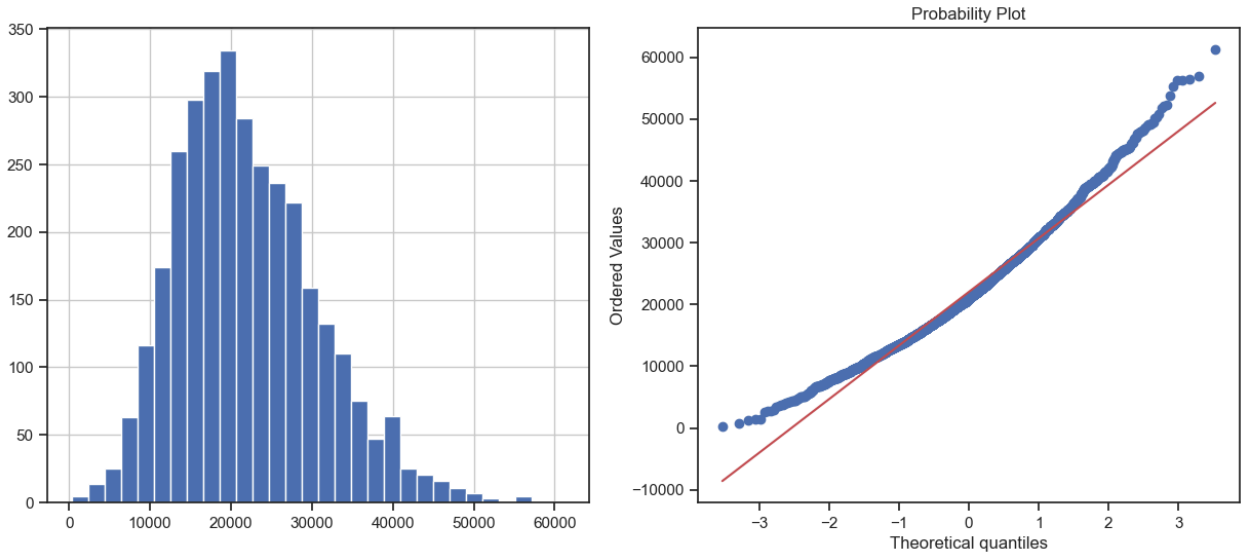
Ввод [34]:
```
data_3 = pd.read_csv('data/water_potability.csv')
```

Ввод [35]:
```
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()
```

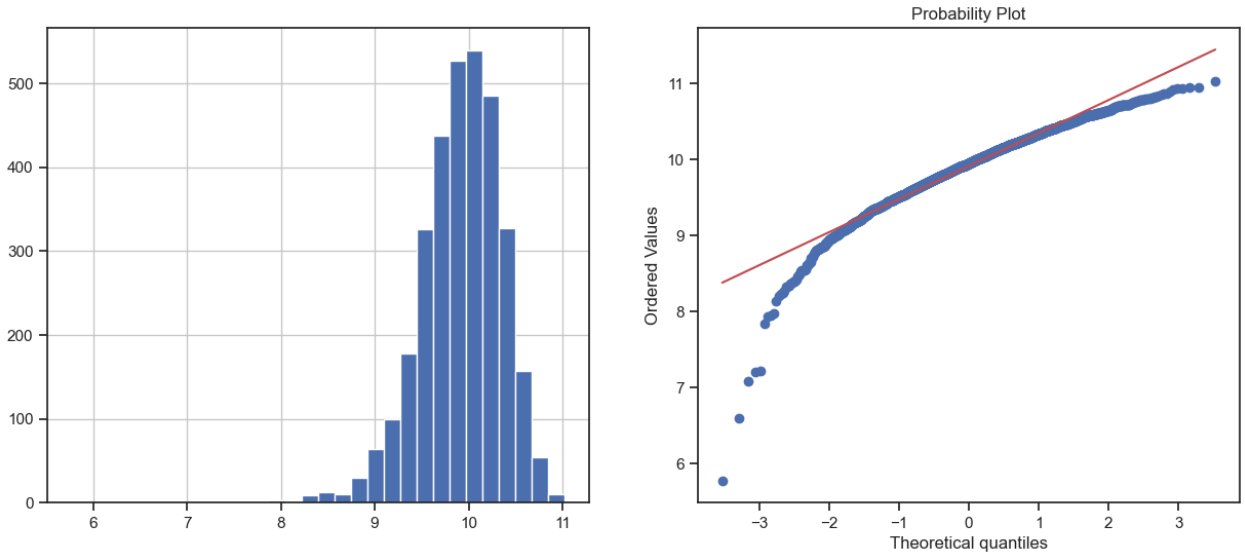
Ввод [36]:
```
data_3.hist(figsize=(20,20))
plt.show()
```



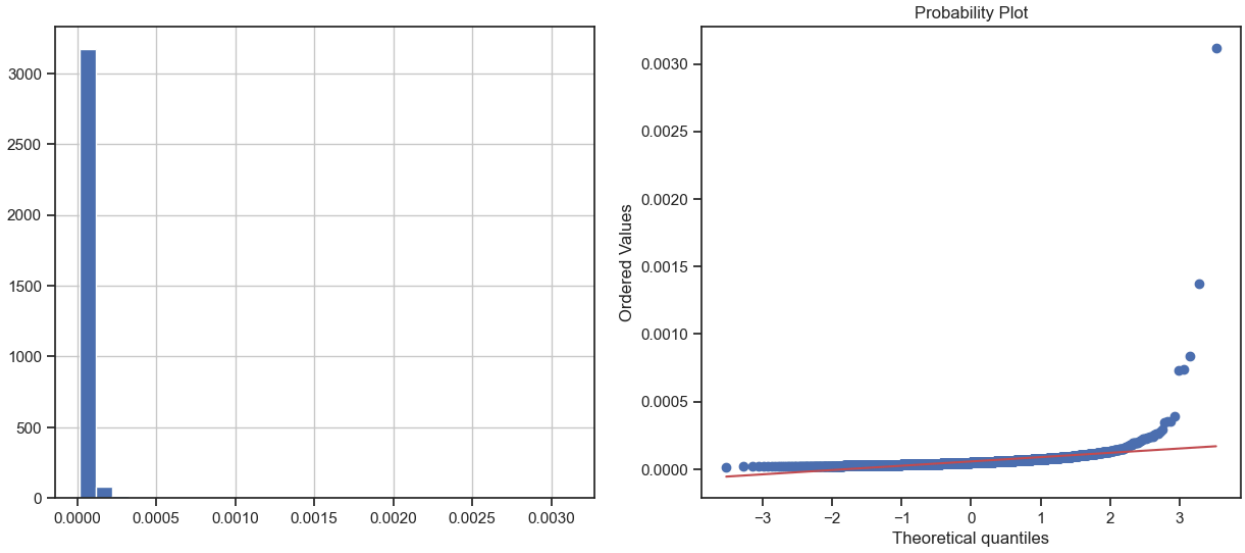
Ввод [37]:
```
# Исходное распределение
diagnostic_plots(data_3, 'Solids')
```



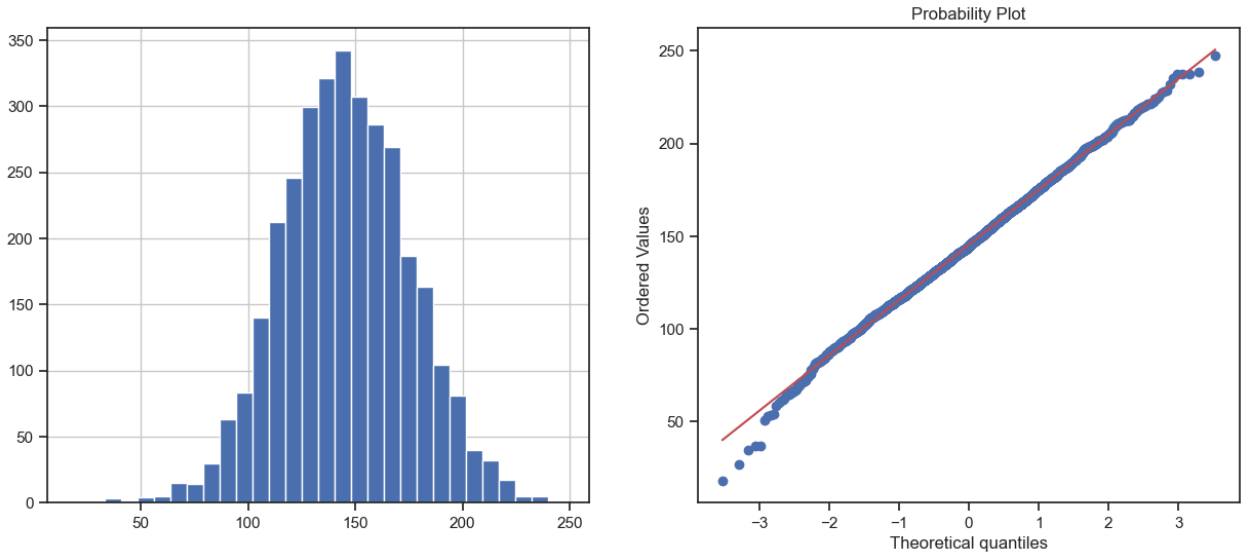
Ввод [38]:
```
# Логарифмическое преобразование
data_3['Solids_log'] = np.log(data_3['Solids'])
diagnostic_plots(data_3, 'Solids_log')
```



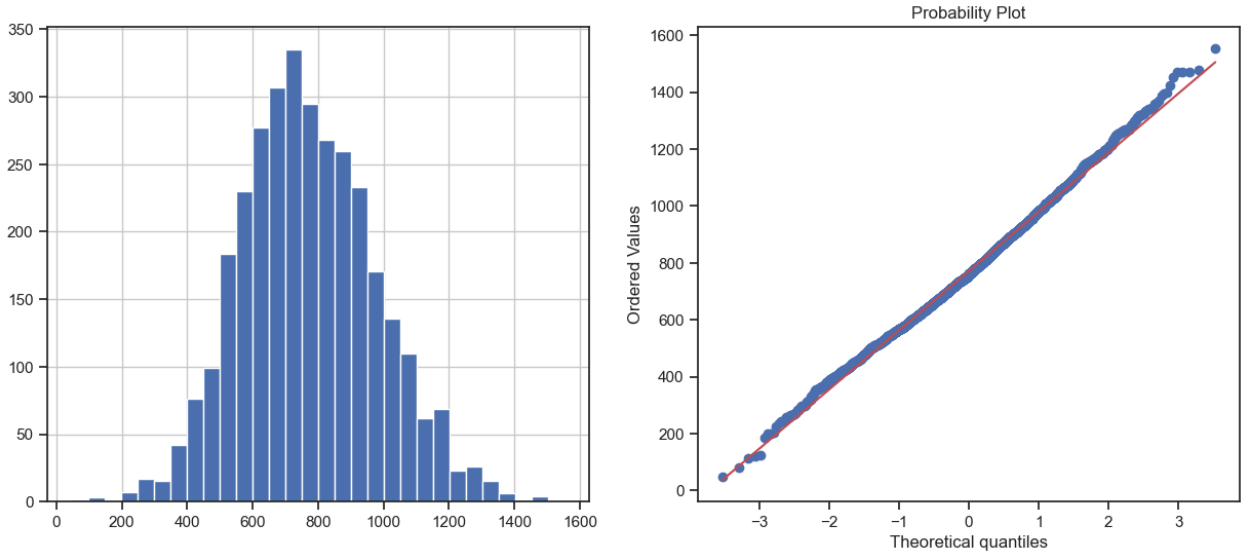
Ввод [39]:
```
# Обратное преобразование
data_3['Solids_reciprocal'] = 1 / (data_3['Solids'])
diagnostic_plots(data_3, 'Solids_reciprocal')
```



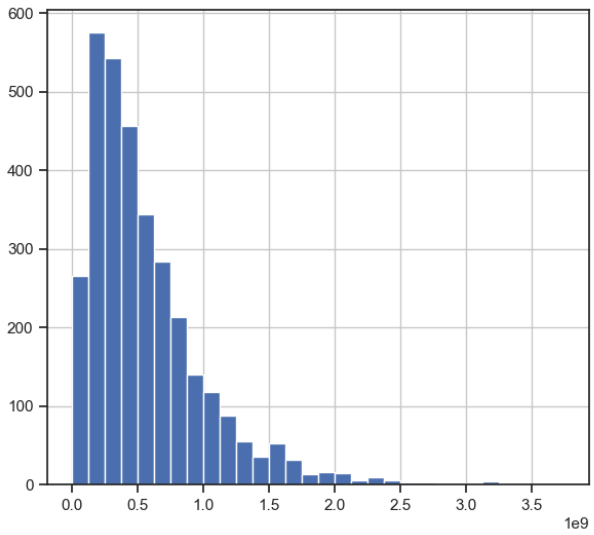
Ввод [40]:
```
# Квадратный корень
data_3['Solids_sqr'] = data_3['Solids']**(1/2)
diagnostic_plots(data_3, 'Solids_sqr')
```

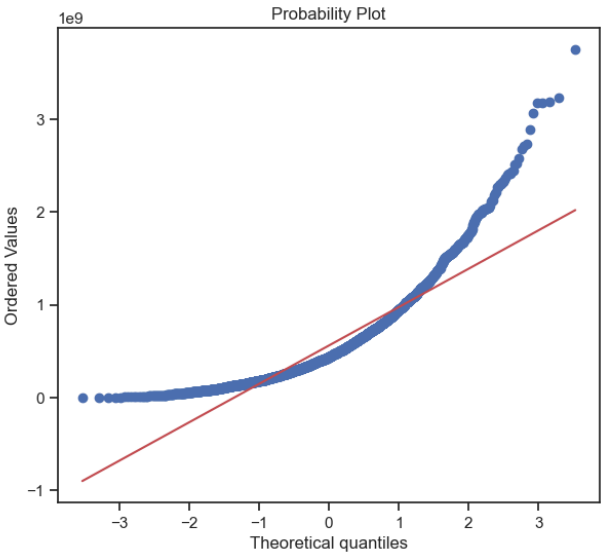


Ввод [41]:
```
# Возведение в степень
data_3['Solids_exp1'] = data_3['Solids']**(1/1.5)
diagnostic_plots(data_3, 'Solids_exp1')
```

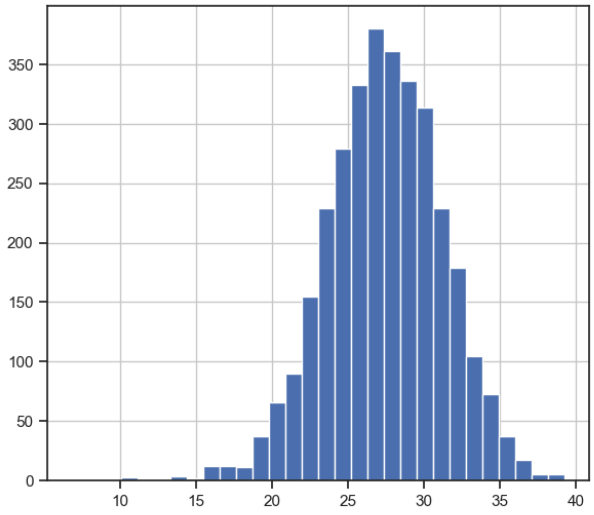


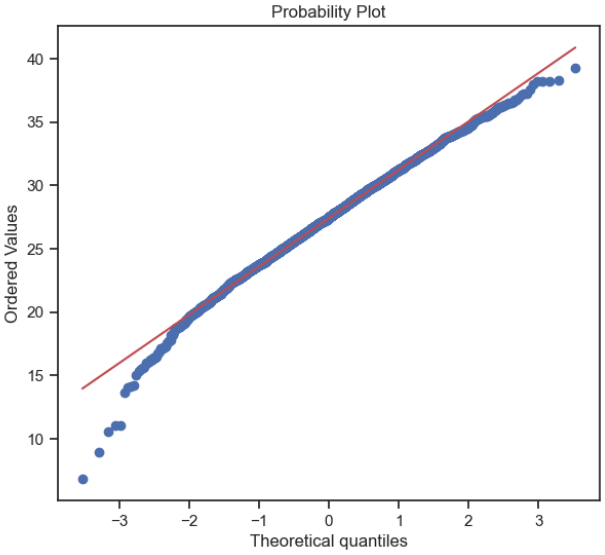
Ввод [42]:
```
data_3['Solids_exp2'] = data_3['Solids']**(2)
diagnostic_plots(data_3, 'Solids_exp2')
```





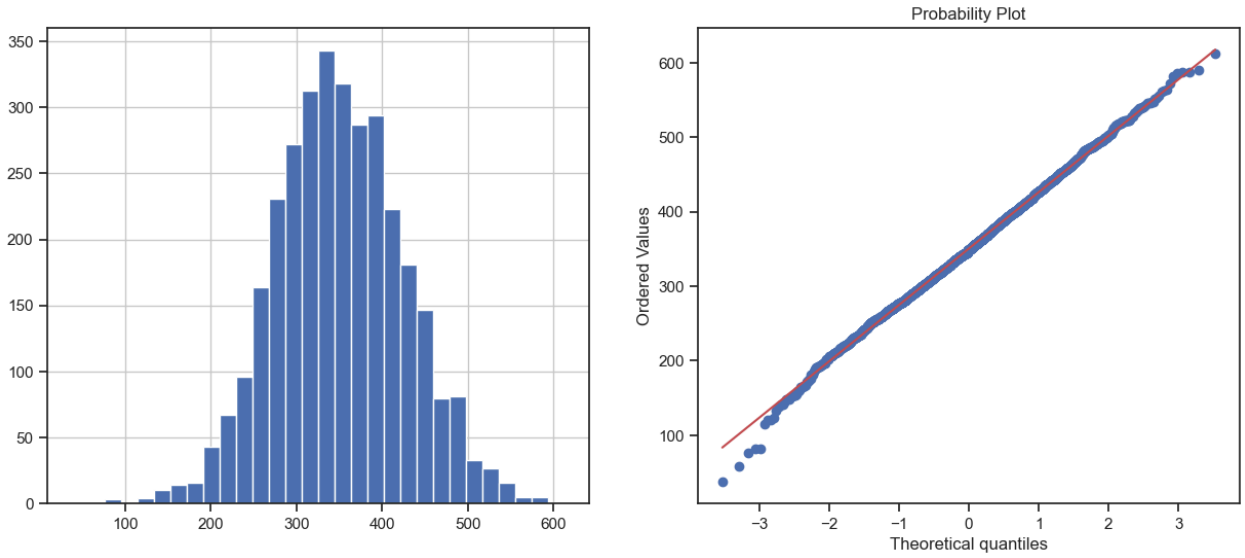
Ввод [43]:
```
data_3['Solids_exp3'] = data_3['Solids']**(0.333)
diagnostic_plots(data_3, 'Solids_exp3')
```





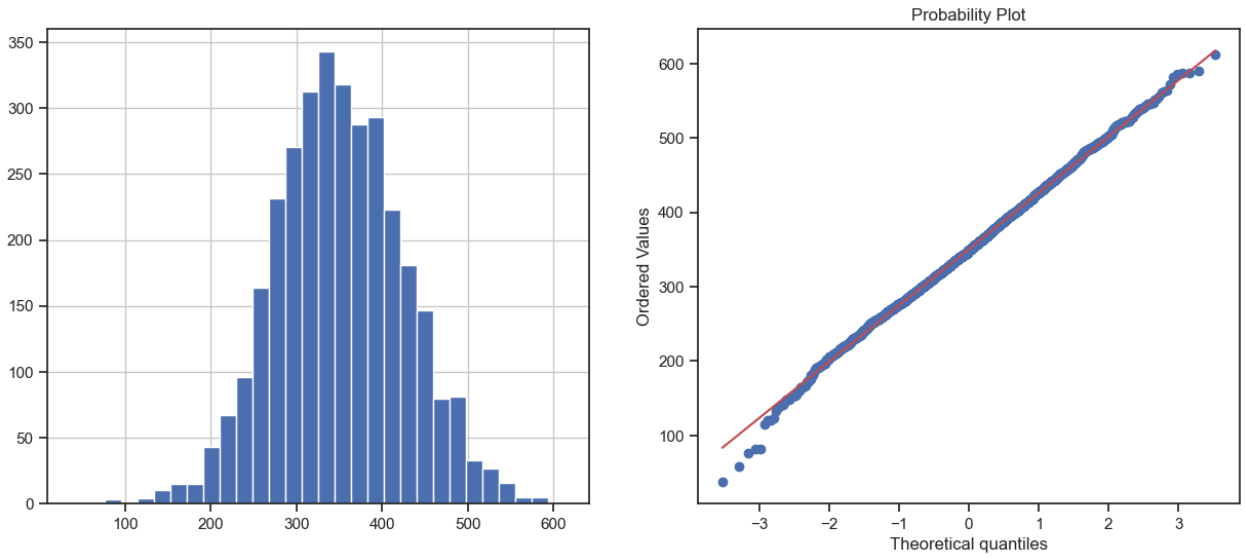
Ввод [44]:
```
# Преобразование Бокса-Кокса
data_3['Solids_boxcox'], param = stats.boxcox(data_3['Solids'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(data_3, 'Solids_boxcox')
```

```
Оптимальное значение λ = 0.5239559434270761
```



Ввод [45]:
```
# Преобразование Йео-Джонсона
data_3['Solids'] = data_3['Solids'].astype('float')
data_3['Solids_yeojohnson'], param = stats.yeojohnson(data_3['Solids'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(data_3, 'Solids_yeojohnson')
```

```
Оптимальное значение λ = 0.5239092055566575
```



Итоги:

В результате нормализации хороший результат получился с использованием преобразований квадратный корень, Бокса-Кокса и Йео-Джонсона.

Ввод [ ]: