Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №2
по дисциплине
«Методы машинного обучения»
на тему

«Обработка признаков»

Выполнил:
студент группы ИУ5-22М
ЧжаоЛян

Москва — 2022 г.

Цель лабораторной работы:

Изучение продвинутых способов предварительной обработки данных для дальнейшего формирования моделей.

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

Загрузка и первичный анализ данных

In [31]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from IPython.display import Image
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
```

In [33]:
```
data = pd.read_csv(r'C:\Users\80667\Desktop\loan_data.csv')
```

In [34]:
```
data.shape
```

Out[34]:
```
(614, 13)
```

In [35]:
```
data.isnull().sum()
```

Out[35]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [42]:
```
data.head(5)
```

Out[42]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0

Обработка пропусков в данных

In [43]:
```
# типы колонок
data.dtypes
```

Out[43]:
```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

In [37]:
```
data_clean = res = data.dropna(axis=1, how='any')
data_clean.shape
```

Out[37]:
```
(614, 6)
```

In [44]:
```
data.isnull().sum()
```

Out[44]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
In [45]: # Колонки с пропусками
         cols_with_na = [c for c in data.columns if data[c].isnull().sum() > 0]
         cols_with_na

Out[45]: ['Gender',
          'Married',
          'Dependents',
          'Self_Employed',
          'LoanAmount',
          'Loan_Amount_Term',
          'Credit_History']
```

```
In [46]: # Доля (процент) пропусков
         [(c, data[c].isnull().mean()) for c in cols_with_na]

Out[46]: [('Gender', 0.021172638436482084),
          ('Married', 0.004885993485342019),
          ('Dependents', 0.024429967426710098),
          ('Self_Employed', 0.05211726384364821),
          ('LoanAmount', 0.035830618892508145),
          ('Loan_Amount_Term', 0.02280130293159609),
          ('Credit_History', 0.08143322475570032)]
```

```
##Датасет достаточно маленький, а доля пропущенных значений
не очень большая. В таком случае можно поработать с
внедрением пропущенных значений.
```

Датасет достаточно маленький, а доля пропущенных значений не очень большая. В таком случае можно поработать с внедрением пропущенных значений.

Заполнение значений для одного признака

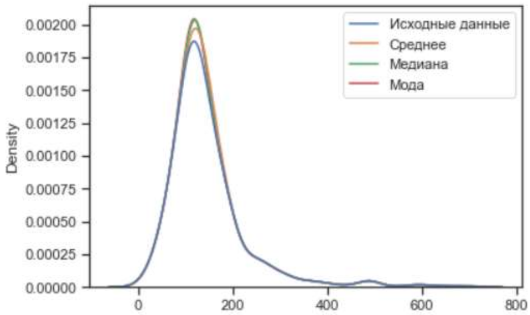
In [47]:
```python
# Пример работы MissingIndicator
temp_x1 = np.array([[np.nan, 1, 3], [np.nan, 0, 5], [3,np.nan, 1]])
print('Исходный массив:')
print(temp_x1)
indicator = MissingIndicator(features='all')
temp_x1_transformed = indicator.fit_transform(temp_x1)
print('Маска пропущенных значений:')
print(temp_x1_transformed)
```

```
Исходный массив:
[[nan  1.   3.]
 [nan  0.   5.]
 [ 3.  nan  1.]]
Маска пропущенных значений:
[[ True False False]
 [ True False False]
 [False  True False]]
```

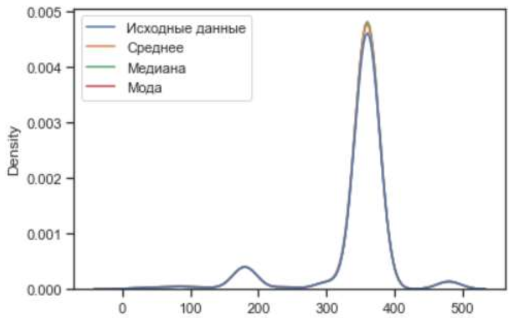
In [48]:
```python
def impute_column(dataset, column, strategy_param, fill_value_param=None):
    """
    Заполнение пропусков в одном признаке
    """
    temp_data = dataset[[column]].values
    size = temp_data.shape[0]

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

    imputer = SimpleImputer(strategy=strategy_param,
                            fill_value=fill_value_param)
    all_data = imputer.fit_transform(temp_data)

    missed_data = temp_data[mask_missing_values_only]
    filled_data = all_data[mask_missing_values_only]

    return all_data.reshape((size,)), filled_data, missed_data
```

In [49]:
```
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)
```

In [50]:
```
research_impute_numeric_column(data, 'LoanAmount')
```



In [51]:
```
research_impute_numeric_column(data, 'Loan_Amount_Term')
```



```
In [53]: data_imp['LoanAmount']=LoanAmount_new
```

```
In [54]: data_imp.isnull().sum()
```

```
Out[54]: Loan_ID              0
         Gender              13
         Married              3
         Dependents          15
         Education            0
         Self_Employed       32
         ApplicantIncome      0
         CoapplicantIncome    0
         LoanAmount           0
         Loan_Amount_Term    14
         Credit_History      50
         Property_Area        0
         Loan_Status          0
         dtype: int64
```

Распределения одномодальные, поэтому можно использовать для импутации моду.

Для категориальных признаков

In [55]:
```
hdata_cat_cols = ['Gender', 'Married', 'Dependents','Self_Employed']
hdata_cat_new = data[hdata_cat_cols].copy(deep=True)
```

In [56]:
```

Gender_cat_new_temp, _, _ = impute_column(hdata_cat_new, 'Gender', 'most_frequent')
Married_cat_new_temp, _, _ = impute_column(hdata_cat_new, 'Married', 'most_frequent')
Dependents_cat_new_temp, _, _ = impute_column(hdata_cat_new, 'Dependents', 'most_frequent')
Self_Employed_cat_new_temp, _, _ = impute_column(hdata_cat_new, 'Self_Employed', 'most_frequent')
```

In [57]:
```
hdata_cat_new['Gender'] = Gender_cat_new_temp
hdata_cat_new['Married'] = Married_cat_new_temp
hdata_cat_new['Dependents'] = Dependents_cat_new_temp
hdata_cat_new['Self_Employed'] = Self_Employed_cat_new_temp
```

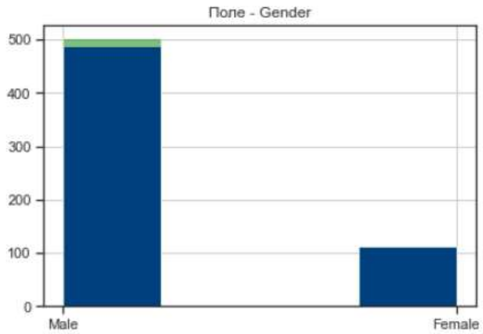
In [58]:
```
data_imp_cat=data.copy()
data_imp_cat['Gender'] = Gender_cat_new_temp
data_imp_cat['Married'] = Married_cat_new_temp
data_imp_cat['Self_Employed'] = Self_Employed_cat_new_temp
```

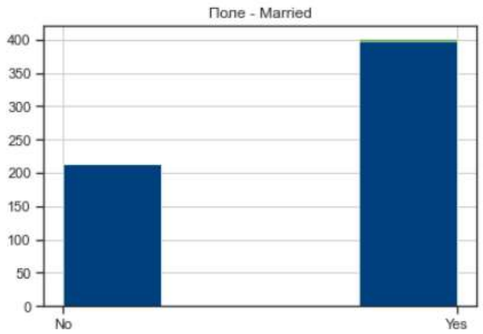
In [59]:
```
data_imp_cat.isnull().sum()
```

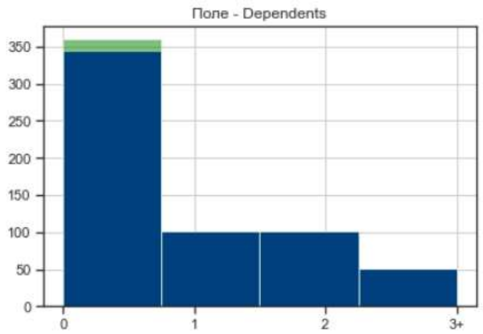
Out[59]:
```
Loan_ID               0
Gender                0
Married               0
Dependents           15
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

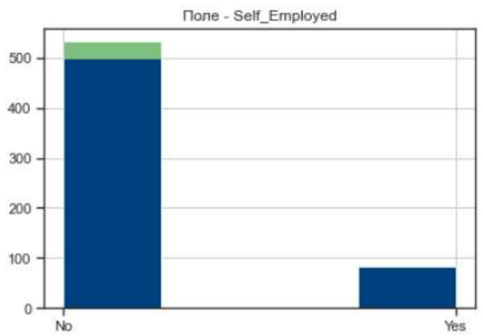
In [60]:
```
def plot_hist_diff(old_ds, new_ds, cols):
    """

    """
    for c in cols:
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.title.set_text('Поле - ' + str(c))
        old_ds[c].hist(bins=4, ax=ax, density=False, color='blue')
        new_ds[c].hist(bins=4, ax=ax, color='green', density=False, alpha=0.5)
        plt.show()
```

In [61]:
```
plot_hist_diff(data, hdata_cat_new, hdata_cat_cols)
```









KNN

In [64]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_knn=data_imp_cat.copy(deep=True) # устранены пропуски строковых признако
data_knn['Gender'] = le.fit_transform(data['Gender'])
data_knn['Married'] = le.fit_transform(data['Married'])
data_knn['Education'] = le.fit_transform(data['Education'])
data_knn['Self_Employed'] = le.fit_transform(data['Self_Employed'])
data_knn['Property_Area'] = le.fit_transform(data['Property_Area'])
data_knn['Dependents'] = le.fit_transform(data['Dependents'])
```

In [65]:
```
knnimpute_cols = ['Dependents', 'LoanAmount','Loan_Amount_Term', 'Credit_History']
```

In [66]:
```
knnimpute_hdata = data_knn[knnimpute_cols].copy()
knnimpute_hdata.head()
```

Out[66]:

	Dependents	LoanAmount	Loan_Amount_Term	Credit_History
0	0	NaN	360.0	1.0
1	1	128.0	360.0	1.0
2	0	66.0	360.0	1.0
3	0	120.0	360.0	1.0
4	0	141.0	360.0	1.0

In [67]:
```
# Признаки с пропусками
knnimpute_hdata.isnull().sum()
```

Out[67]:
```
Dependents           0
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
dtype: int64
```

In [68]:
```
knnimputer = KNNImputer(
    n_neighbors=5,
    weights='distance',
    metric='nan_euclidean',
    add_indicator=False,
)
knnimpute_hdata_imputed_temp = knnimputer.fit_transform(knnimpute_hdata)
knnimpute_hdata_imputed = pd.DataFrame(knnimpute_hdata_imputed_temp, columns=knnimpute_hdata.columns)
knnimpute_hdata_imputed.head()
```

Out[68]:

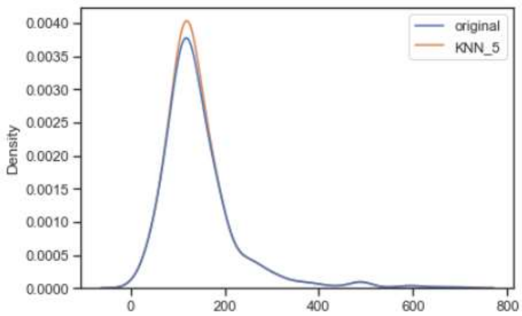
	Dependents	LoanAmount	Loan_Amount_Term	Credit_History
0	0.0	120.8	360.0	1.0
1	1.0	128.0	360.0	1.0
2	0.0	66.0	360.0	1.0
3	0.0	120.0	360.0	1.0
4	0.0	141.0	360.0	1.0

In [69]:
```
# Пропуски заполнены
knnimpute_hdata_imputed.isnull().sum()
```

Out[69]:
```
Dependents          0
LoanAmount          0
Loan_Amount_Term    0
Credit_History      0
dtype: int64
```

In [70]:
```
LoanAmount_df = pd.DataFrame({'original': knnimpute_hdata['LoanAmount'].values})
LoanAmount_df['KNN_5'] = knnimpute_hdata_imputed['LoanAmount']
sns.kdeplot(data=LoanAmount_df)
```

Out[70]: <AxesSubplot:ylabel='Density'>



кодирование категориальных признаков

In [71]:
```
data_=data.copy(deep=True)
LoanAmount_new, _,_ = impute_column(data_, 'LoanAmount', 'most_frequent')
Loan_Amount_Term_new, _,_ = impute_column(data_, 'Loan_Amount_Term', 'most_frequent')
Credit_History_new, _,_ = impute_column(data_, 'Credit_History', 'most_frequent')

Gender_cat_new_temp, _, _ = impute_column(data_, 'Gender', 'most_frequent')
Married_cat_new_temp, _, _ = impute_column(data_, 'Married', 'most_frequent')
Education_cat_new_temp, _, _ = impute_column(data_, 'Education', 'most_frequent')
Self_Employed_cat_new_temp, _, _ = impute_column(data_, 'Self_Employed', 'most_frequent')
Property_Area_cat_new_temp, _, _ = impute_column(data_, 'Property_Area', 'most_frequent')
Dependents_cat_new_temp, _, _ = impute_column(data_, 'Dependents', 'most_frequent')
```

In [72]:
```
data_['LoanAmount'] = LoanAmount_new
data_['Loan_Amount_Term']=Loan_Amount_Term_new
data_['Credit_History']=Credit_History_new
data_['Gender'] = Gender_cat_new_temp
data_['Married'] = Married_cat_new_temp
data_['Education'] = Education_cat_new_temp
data_['Self_Employed'] = Self_Employed_cat_new_temp
data_['Property_Area'] = Property_Area_cat_new_temp
data_['Dependents'] = Dependents_cat_new_temp
```

In [73]:
```
data_=data_.drop(['Loan_ID'], axis=1)
data_['Loan_Status'] = le.fit_transform(data_['Loan_Status'])
```

label encoding

In [74]:
```
data_.head()
```

Out[74]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	Male	No	0	Graduate	No	5849	0.0	120.0	3
1	Male	Yes	1	Graduate	No	4583	1508.0	128.0	3
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	3
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	3
4	Male	No	0	Graduate	No	6000	0.0	141.0	3

In [75]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_Lable_enc=data_.copy(deep=True)
data_Lable_enc['Gender'] = le.fit_transform(data_['Gender'])
print(data_Lable_enc['Gender'].unique())
print(le.inverse_transform([1,0]))
data_Lable_enc['Married'] = le.fit_transform(data_Lable_enc['Married'])
data_Lable_enc['Education'] = le.fit_transform(data_Lable_enc['Education'])
data_Lable_enc['Self_Employed'] = le.fit_transform(data_Lable_enc['Self_Employed'])
data_Lable_enc['Property_Area'] = le.fit_transform(data_Lable_enc['Property_Area'])
```

```
[1 0]
['Male' 'Female']
```

In [76]:
```
data_Lable_enc.head()
```

Out[76]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	1	0	0	0	0	5849	0.0	120.0	3
1	1	1	1	0	0	4583	1508.0	128.0	3
2	1	1	0	0	1	3000	0.0	66.0	3
3	1	1	0	1	0	2583	2358.0	120.0	3
4	1	0	0	0	0	6000	0.0	141.0	3

one-hot encoding

In [88]:
```
data_dumm=data_.copy(deep=True)
```

In [89]:
```
# Добавление отдельной колонки, признака пустых значений
pd.get_dummies(data_dumm[['Gender']], dummy_na=True).head()
```

Out[89]:

	Gender_Female	Gender_Male	Gender_nan
0	0	1	0
1	0	1	0
2	0	1	0
3	0	1	0
4	0	1	0

In [90]:
```
def encode_and_bind(original_dataframe, feature_to_encode):
    dummies = pd.get_dummies(original_dataframe[[feature_to_encode]])
    original_dataframe = pd.concat([original_dataframe, dummies], axis=1)
    original_dataframe = original_dataframe.drop(feature_to_encode,axis = 1)
    return original_dataframe
```

In [91]:
```
data_dumm=encode_and_bind(data_dumm, 'Gender')
```

In [92]:
```
data_dumm.head()
```

Out[92]:

	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Cre
0	No	0	Graduate	No	5849	0.0	120.0	360.0	
1	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	
2	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	
3	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	
4	No	0	Graduate	No	6000	0.0	141.0	360.0	

In [100]:
```
data=data_.copy(deep=True)
```

In [113]:
```
pip install category-encoders
```

```
Collecting category-encoders
  Downloading category_encoders-2.5.0-py2.py3-none-any.whl (69 kB)
Requirement already satisfied: scipy>=1.0.0 in c:\zl\work\anaconda\anaconda\lib\site-packages (from category-encoders) (1.7.3)
Requirement already satisfied: statsmodels>=0.9.0 in c:\zl\work\anaconda\anaconda\lib\site-packages (from category-encoders) (0.12.2)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\zl\work\anaconda\anaconda\lib\site-packages (from category-encoders) (0.24.2)
Requirement already satisfied: patsy>=0.5.1 in c:\zl\work\anaconda\anaconda\lib\site-packages (from category-encoders) (0.5.2)
Requirement already satisfied: pandas>=1.0.5 in c:\zl\work\anaconda\anaconda\lib\site-packages (from category-encoders) (1.3.4)
Requirement already satisfied: numpy>=1.14.0 in c:\zl\work\anaconda\anaconda\lib\site-packages (from category-encoders) (1.22.3)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: pytz>=2017.3 in c:\zl\work\anaconda\anaconda\lib\site-packages (from pandas>=1.0.5->category-encoders) (2021.
3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\zl\work\anaconda\anaconda\lib\site-packages (from pandas>=1.0.5->category-encode
rs) (2.8.2)
Requirement already satisfied: six in c:\zl\work\anaconda\anaconda\lib\site-packages (from patsy>=0.5.1->category-encoders) (1.16.0)
Requirement already satisfied: joblib>=0.11 in c:\zl\work\anaconda\anaconda\lib\site-packages (from scikit-learn>=0.20.0->category-encoders)
(1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\zl\work\anaconda\anaconda\lib\site-packages (from scikit-learn>=0.20.0->category-e
ncoders) (2.2.0)
Installing collected packages: category-encoders
Successfully installed category-encoders-2.5.0
```

In [114]:
```
from category_encoders.one_hot import OneHotEncoder as ce_OneHotEncoder
```

In [115]:
```
ce_OneHotEncoder1 = ce_OneHotEncoder()
data_OHE = ce_OneHotEncoder1.fit_transform(data_[data_.columns.difference(['Loan_Status'])])
```

In [116]:
```
data_OHE
```

Out[116]:

	ApplicantIncome	CoapplicantIncome	Credit_History	Dependents_1	Dependents_2	Dependents_3	Dependents_4	Education_1	Education_2	Gender_1	Ge
0	5849	0.0	1.0	1	0	0	0	1	0	1	
1	4583	1508.0	1.0	0	1	0	0	1	0	1	
2	3000	0.0	1.0	1	0	0	0	1	0	1	
3	2583	2358.0	1.0	1	0	0	0	0	1	1	
4	6000	0.0	1.0	1	0	0	0	1	0	1	
...	...	...	...	...	...	...	...	...	...	...	
609	2900	0.0	1.0	1	0	0	0	1	0	0	
610	4106	0.0	1.0	0	0	0	1	1	0	1	
611	8072	240.0	1.0	0	1	0	0	1	0	1	
612	7583	0.0	1.0	0	0	1	0	1	0	1	
613	4583	0.0	0.0	1	0	0	0	1	0	0	

614 rows × 20 columns

Count (frequency) encoding

In [117]:
```
from category_encoders.count import CountEncoder as ce_CountEncoder
```

In [118]:
```
data_
```

Out[118]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_
0	Male	No	0	Graduate	No	5849	0.0	120.0	360.0	1.0	U
1	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	U
3	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	U
4	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	U
...	...	...	...	...	...	...	...	...	...	...	
609	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	
610	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	
611	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	U
612	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	U
613	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semi

614 rows × 12 columns

In [119]:
```
ce_CountEncoder1 = ce_CountEncoder()
data_COUNT_ENC = ce_CountEncoder1.fit_transform(data_[data_.columns.difference(['Loan_Status'])])
```

In [120]:
```
data_COUNT_ENC
```

Out[120]:

	ApplicantIncome	CoapplicantIncome	Credit_History	Dependents	Education	Gender	LoanAmount	Loan_Amount_Term	Married	Property_Area	Self_Empl
0	5849	0.0	1.0	360	480	502	120.0	360.0	213	202	
1	4583	1508.0	1.0	102	480	502	128.0	360.0	401	179	
2	3000	0.0	1.0	360	480	502	66.0	360.0	401	202	
3	2583	2358.0	1.0	360	134	502	120.0	360.0	401	202	
4	6000	0.0	1.0	360	480	502	141.0	360.0	213	202	
...	...	...	...	...	...	...	...	...	...	...	
609	2900	0.0	1.0	360	480	112	71.0	360.0	213	179	
610	4106	0.0	1.0	51	480	502	40.0	180.0	401	179	
611	8072	240.0	1.0	102	480	502	253.0	360.0	401	202	
612	7583	0.0	1.0	101	480	502	187.0	360.0	401	202	
613	4583	0.0	0.0	360	480	112	133.0	360.0	213	233	

614 rows × 11 columns

In [121]:
```
data_['Property_Area'].unique()
```

Out[121]:
```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [122]:
```
data_COUNT_ENC['Property_Area'].unique()
```

Out[122]:
```
array([202, 179, 233], dtype=int64)
```

Target (Mean) encoding

In [123]:
```python
# На самом деле этот метод реализует Mean encoding
from category_encoders.target_encoder import TargetEncoder as ce_TargetEncoder
```

In [124]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
#data_6=data.copy(deep=True)
#data_6['Loan_Status'] = le.fit_transform(data_6['Loan_Status'])
```

In [127]:
```python
ce_TargetEncoder1 = ce_TargetEncoder()
data_MEAN_ENC = ce_TargetEncoder1.fit_transform(data_[data_.columns.difference(['Loan_Status'])], data_['Loan_Status'])
```

```
C:\ZL\Work\Anaconda\Anaconda\lib\site-packages\category_encoders\target_encoder.py:92: FutureWarning: Default parameter min_samples
_leaf will change in version 2.6.See https://github.com/scikit-learn-contrib/category_encoders/issues/327
  warnings.warn("Default parameter min_samples_leaf will change in version 2.6."
C:\ZL\Work\Anaconda\Anaconda\lib\site-packages\category_encoders\target_encoder.py:97: FutureWarning: Default parameter smoothing w
ill change in version 2.6.See https://github.com/scikit-learn-contrib/category_encoders/issues/327
  warnings.warn("Default parameter smoothing will change in version 2.6."
```

In [128]:
```python
data_MEAN_ENC
```

Out[128]:

	ApplicantIncome	CoapplicantIncome	Credit_History	Dependents	Education	Gender	LoanAmount	Loan_Amount_Term	Married	Property_Area	Self_En
0	5849	0.0	1.0	0.686111	0.708333	0.691235	120.0	360.0	0.629108	0.658416	0
1	4583	1508.0	1.0	0.647059	0.708333	0.691235	128.0	360.0	0.718204	0.614525	0
2	3000	0.0	1.0	0.686111	0.708333	0.691235	66.0	360.0	0.718204	0.658416	0
3	2583	2358.0	1.0	0.686111	0.611940	0.691235	120.0	360.0	0.718204	0.658416	0
4	6000	0.0	1.0	0.686111	0.708333	0.691235	141.0	360.0	0.629108	0.658416	0
...	...	...	...	...	...	...	...	...	...	...	
609	2900	0.0	1.0	0.686111	0.708333	0.669643	71.0	360.0	0.629108	0.614525	0
610	4106	0.0	1.0	0.647059	0.708333	0.691235	40.0	180.0	0.718204	0.614525	0
611	8072	240.0	1.0	0.647059	0.708333	0.691235	253.0	360.0	0.718204	0.658416	0
612	7583	0.0	1.0	0.752475	0.708333	0.691235	187.0	360.0	0.718204	0.658416	0
613	4583	0.0	0.0	0.686111	0.708333	0.669643	133.0	360.0	0.629108	0.768240	0

614 rows × 11 columns

In [129]:
```python
data_['Property_Area'].unique()
```

Out[129]:
```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [130]:
```python
data_MEAN_ENC['Property_Area'].unique()
```

Out[130]:
```
array([0.65841584, 0.61452514, 0.76824034])
```

In [131]:
```python
def check_mean_encoding(field):
    for s in data[field].unique():
        data_filter = data_[data_[field]==s]
        if data_filter.shape[0] > 0:
            prob = sum(data_filter['Loan_Status']) / data_filter.shape[0]
            print(s, '-' , prob)
```

In [132]:
```python
check_mean_encoding('Property_Area')
```

```
Urban - 0.6584158415841584
Rural - 0.6145251396648045
Semiurban - 0.7682403433476395
```

Weight of evidence (WoE) encoding

In [133]:
```
from category_encoders.woe import WOEEncoder as ce_WOEEncoder
```

In [134]:
```
ce_WOEEncoder1 = ce_WOEEncoder()
data_WOE_ENC = ce_WOEEncoder1.fit_transform(data_[data_.columns.difference(['Loan_Status'])], data_['Loan_Status'])
```

In [135]:
```
data_WOE_ENC
```

Out[135]:

	ApplicantIncome	CoapplicantIncome	Credit_History	Dependents	Education	Gender	LoanAmount	Loan_Amount_Term	Married	Property_Area	Self_E
0	5849	0.0	1.0	-0.004645	0.101247	0.020471	120.0	360.0	-0.258627	-0.132531	
1	4583	1508.0	1.0	-0.188101	0.101247	0.020471	128.0	360.0	0.148353	-0.320840	
2	3000	0.0	1.0	-0.004645	0.101247	0.020471	66.0	360.0	0.148353	-0.132531	-
3	2583	2358.0	1.0	-0.004645	-0.333327	0.020471	120.0	360.0	0.148353	-0.132531	
4	6000	0.0	1.0	-0.004645	0.101247	0.020471	141.0	360.0	-0.258627	-0.132531	
...	...	...	...	...	...	...	...	...	...	...	
609	2900	0.0	1.0	-0.004645	0.101247	-0.088728	71.0	360.0	-0.258627	-0.320840	
610	4106	0.0	1.0	-0.199954	0.101247	0.020471	40.0	180.0	0.148353	-0.320840	
611	8072	240.0	1.0	-0.188101	0.101247	0.020471	253.0	360.0	0.148353	-0.132531	
612	7583	0.0	1.0	0.303834	0.101247	0.020471	187.0	360.0	0.148353	-0.132531	
613	4583	0.0	0.0	-0.004645	0.101247	-0.088728	133.0	360.0	-0.258627	0.403748	-

614 rows × 11 columns

In [136]:
```
# Проверка для поля "Пол"
data_['Property_Area'].unique()
```

Out[136]:
```
array(['Urban', 'Rural', 'Semiurban'], dtype=object)
```

In [137]:
```

data_WOE_ENC['Property_Area'].unique()
```

Out[137]:
```
array([-0.13253074, -0.32084034,  0.40374837])
```

In [138]:
```
class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.3, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

In [139]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
```

In [140]:
```
clas_models_dict = {'LogR': LogisticRegression(max_iter=1000),
                    'KNN_5':KNeighborsClassifier(n_neighbors=5),
                    'Tree':DecisionTreeClassifier(),
                    'GB': GradientBoostingClassifier(),
                    'RF':RandomForestClassifier(n_estimators=50, random_state=1, max_depth=3)}
```

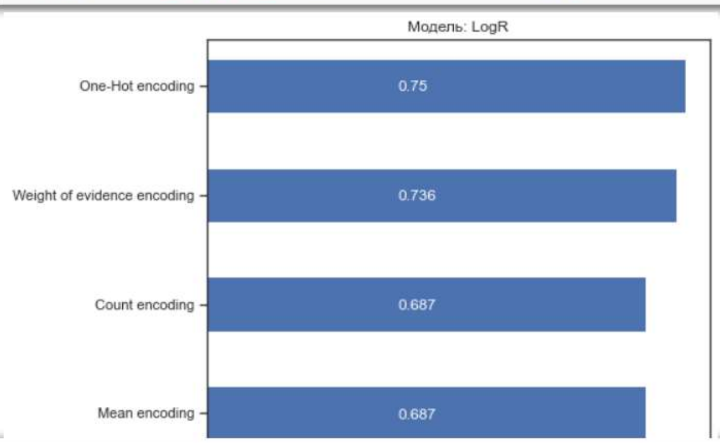
In [141]:
```
X_data_dict = {
               'One-Hot encoding': data_OHE,
               'Count encoding': data_COUNT_ENC,
               'Mean encoding': data_MEAN_ENC,
               'Weight of evidence encoding': data_WOE_ENC,

              }
```

In [142]:
```
def test_models(clas_models_dict, X_data_dict, y_data):

    logger = MetricLogger()

    for model_name, model in clas_models_dict.items():

        for data_name, X_data in X_data_dict.items():

            X_train, X_test, y_train, y_test = train_test_split(
                X_data, y_data, test_size=0.3, random_state=1)

            model.fit(X_train, y_train)
            pred1 = model.predict_proba(X_train)
            pred2 = model.predict_proba(X_test)
            roc_auc = roc_auc_score(y_test, pred2[:, 1])
            logger.add(model_name, data_name, roc_auc)

    return logger
```

In [143]:
```
%%time
logger = test_models(clas_models_dict, X_data_dict, data['Loan_Status'])
```

```
Wall time: 642 ms
```

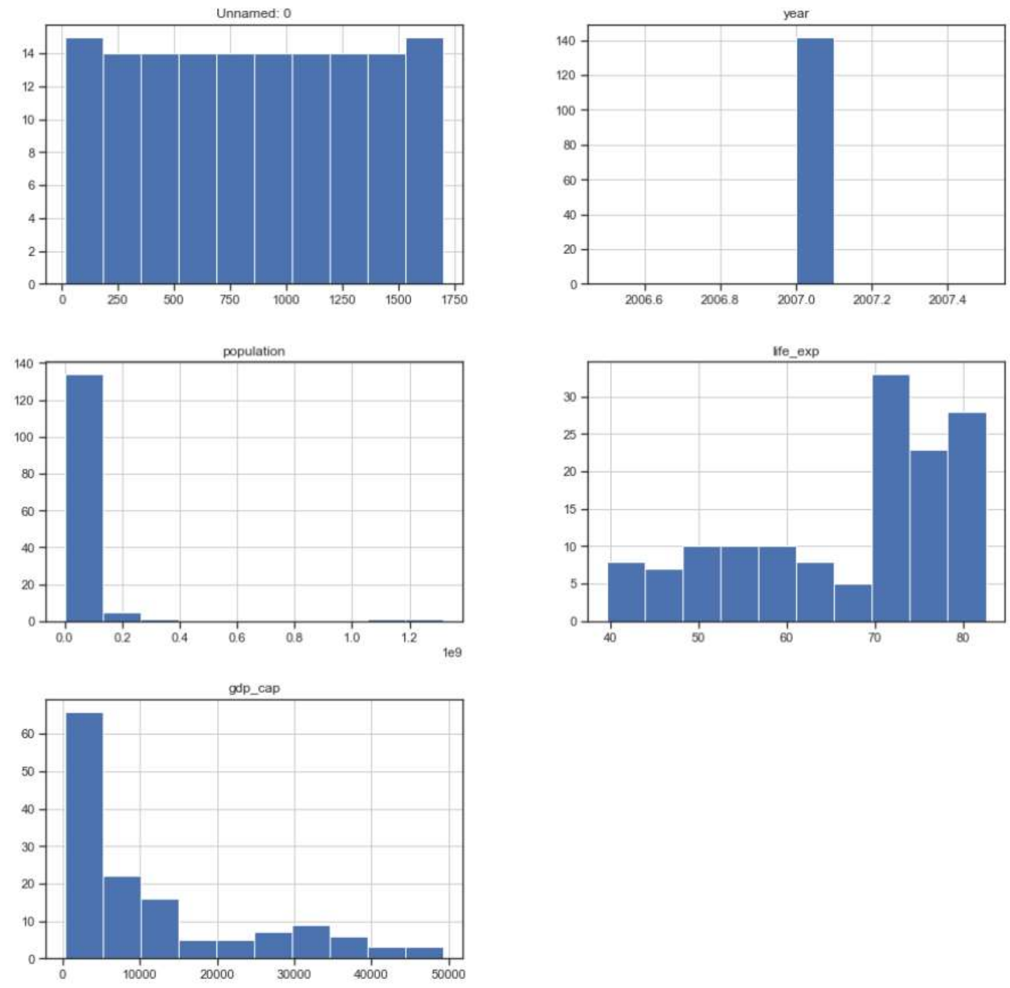
In [144]:
```
# Построим графики метрик качества модели
for model in clas_models_dict:
    logger.plot('Модель: ' + model, model, figsize=(7, 6))
```



нормализация числовых признаков

In [145]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
```

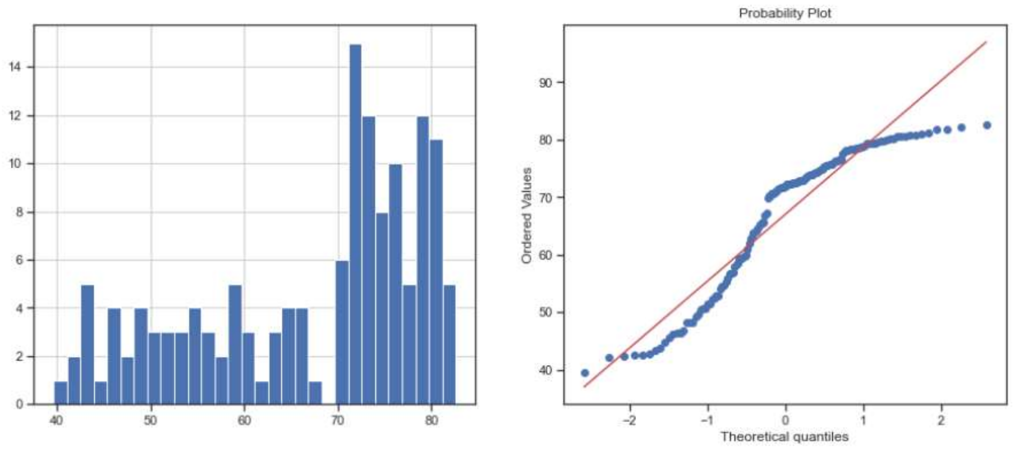
In [151]:
```
data = pd.read_csv(r"C:\Users\80667\Desktop\word.csv")
data.hist(figsize=(15,15))
plt.show()
```



In [152]:
```
data.head()
```
Out[152]:

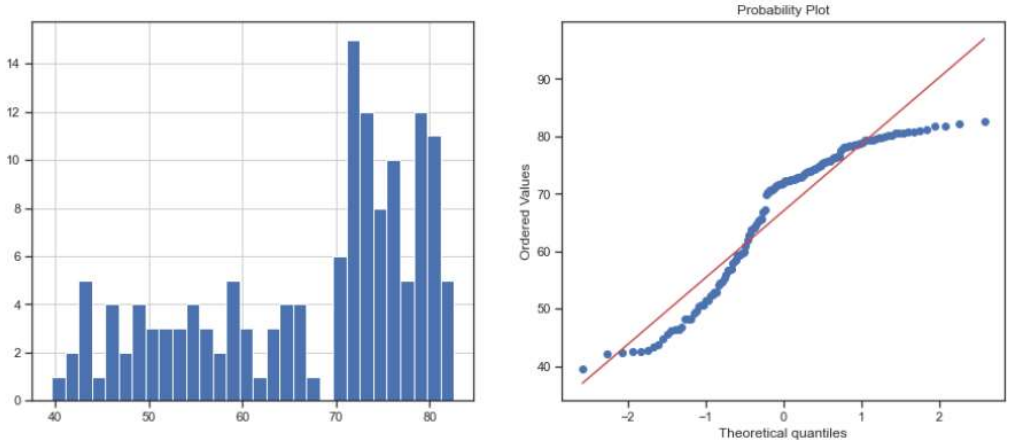
	Unnamed: 0	country	year	population	continent	life_exp	gdp_cap
0	11	Afghanistan	2007	31889923	Asia	43.828	974.580338
1	23	Albania	2007	3600523	Europe	76.423	5937.029526
2	35	Algeria	2007	33333216	Africa	72.301	6223.367465
3	47	Angola	2007	12420476	Africa	42.731	4797.231267
4	59	Argentina	2007	40301927	Americas	75.320	12779.379640

In [153]:
```
data.shape
```
Out[153]: (142, 7)

In [155]:
```
data = data.dropna(subset=['life_exp'],axis=0)
```
In [156]:
```
data = data[data['life_exp'] != 0]
```
In [157]:
```
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()
```
In [158]:
```
diagnostic_plots(data, 'life_exp')
```


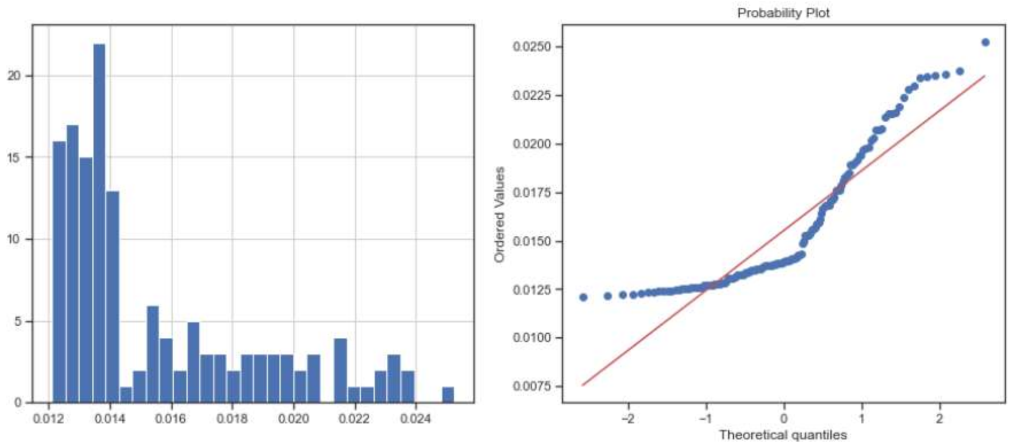
Логарифмическое преобразование

In [159]:
```
data['Perceptions of corruption log'] = np.log(data['life_exp'])
diagnostic_plots(data, 'life_exp')
```



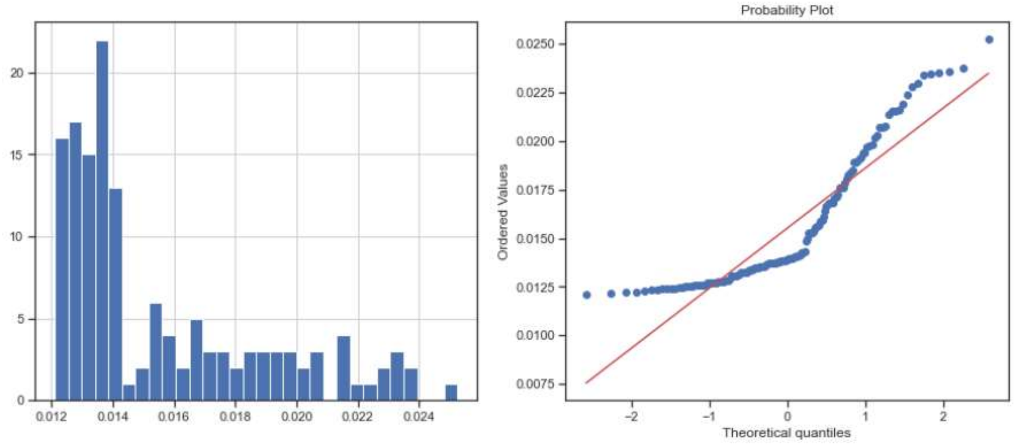

```
#Обратное преобразование
```

In [160]:
```
data['life_exp'] = 1 / (data['life_exp'])
diagnostic_plots(data, 'life_exp')
```



Обратное преобразование

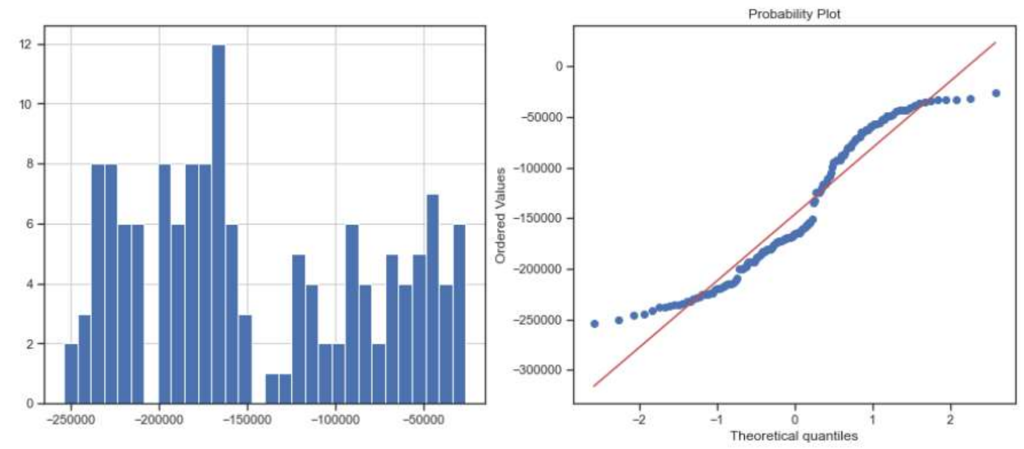
In [160]:
```
data['life_exp'] = 1 / (data['life_exp'])
diagnostic_plots(data, 'life_exp')
```



Преобразование Бокса-Кокса

In [161]:
```
data['life_exp boxcox'], param = stats.boxcox(data['life_exp'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(data, 'life_exp boxcox')
```

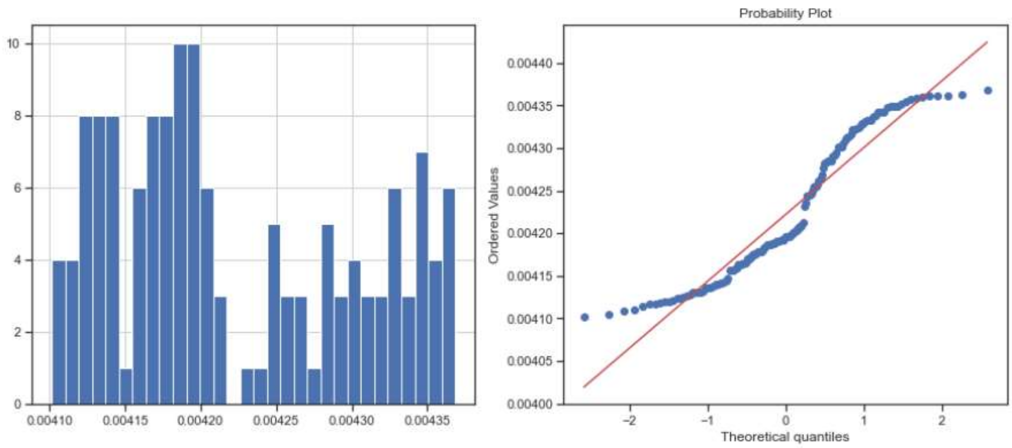
```
Оптимальное значение λ = -3.073480377920155
```



Преобразование Йео-Джонсона

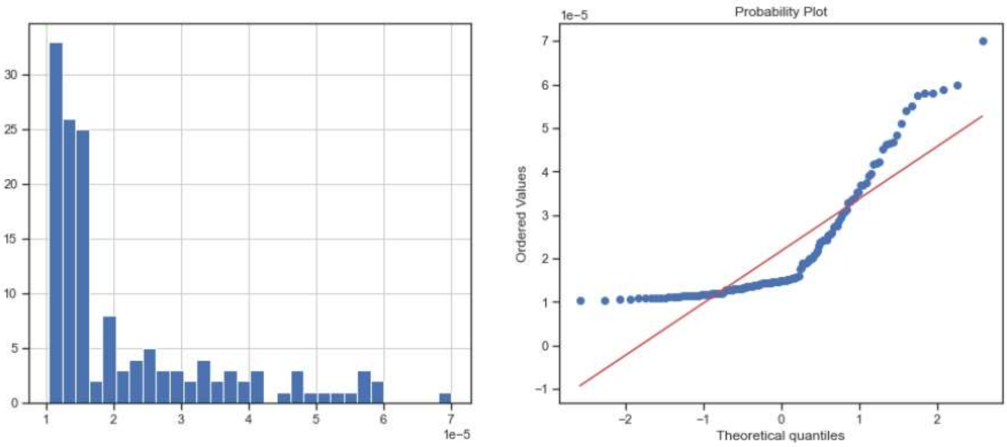
In [162]:
```
# Необходимо преобразовать данные к действительному типу

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

```
Оптимальное значение λ = -228.14980130334024
```



Возведение в степень

In [163]:
```
data['life_exp exp'] = data['life_exp']**(2.6)
diagnostic_plots(data, 'life_exp exp')
```



In [ ]:

Список литературы

[1] Гапанюк Ю. Е. Лабораторная работа «Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных» [Электронный ресурс] // GitHub. — 2019. — Режим доступа: https://github.com/ugapanyuk/ml_course/wiki/LAB_MISSING (дата обращения: 05.04.2019).
[2] Team The IPython Development. IPython 7.3.0 Documentation [Electronic resource]//Read the Docs. — 2019. — Access mode: https://ipython.readthedocs.io/en/stable/ (online; accessed: 20.02.2019).
[3] Waskom M. seaborn 0.9.0 documentation [Electronic resource] // PyData. — 2018. — Access mode: https://seaborn.pydata.org/ (online; accessed: 20.02.2019).
[4] pandas 0.24.1 documentation [Electronic resource] // PyData. — 2019. — Access mode: http://pandas.pydata.org/pandas-docs/stable/ (online; accessed: 20.02.2019).
[5] Gupta L. Google Play Store Apps [Electronic resource] // Kaggle. — 2019. — Access mode: https://www.kaggle.com/lava18/google-play-store-apps (online; accessed:05.04.2019).