МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

Лабораторная работа №2
по курсу «Методы машинного обучения»

ИСПОЛНИТЕЛЬНИЦА: Абросимова Н.Г.
ИУ5-24М

_____
подпись

"__"_____2021 г.

ПРЕПОДАВАТЕЛЬ: _____
ФИО

_____
подпись

"__"_____2021 г.

Москва - 2021

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.

Текст программы и экранные формы

```
import numpy as np
 import pandas as pd
 import seaborn as sns
 import matplotlib.pyplot as plt
 %matplotlib inline
 sns.set(style="ticks")
data=pd.read_csv('oec.csv', sep=",")
```

```
#размер датасета
data.shape
```

Out[3]:

```
(3584, 25)
```

```
data.head()
```

Out[4]:

	PlanetIdentifier	TypeFlag	PlanetaryMassJpt	RadiusJpt	PeriodDays	SemiMajorAxisAU	Eccentricity	PeriastronDeg	LongitudeDeg	AscendingNodeDeg
0	HD 143761 b	0	1.0450	NaN	39.845800	0.2196	0.037	270.6	NaN	NaN
1	HD 143761 c	0	0.0790	NaN	102.540000	0.4123	0.050	190.0	NaN	NaN
2	KOI-1843.03	0	0.0014	0.054	0.176891	0.0048	NaN	NaN	NaN	NaN
3	KOI-1843.01	0	NaN	0.114	4.194525	0.0390	NaN	NaN	NaN	NaN
4	KOI-1843.02	0	NaN	0.071	6.356006	0.0520	NaN	NaN	NaN	NaN

5 rows × 25 columns

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

Out[5]:

```
['PlanetaryMassJpt',
 'RadiusJpt',
 'PeriodDays',
 'SemiMajorAxisAU',
 'Eccentricity',
 'PeriastronDeg',
 'LongitudeDeg',
 'AscendingNodeDeg',
 'InclinationDeg',
 'SurfaceTempK',
 'AgeGyr',
 'DiscoveryMethod',
 'DiscoveryYear',
 'LastUpdated',
```

```
 'RightAscension',
 'Declination',
 'DistFromSunParsec',
 'HostStarMassSlrMass',
 'HostStarRadiusSlrRad',
 'HostStarMetallicity',
 'HostStarTempK',
 'HostStarAgeGyr']
```

```
# Количество пропусков [(c, data[c].isnull().sum()) for c in
hcols_with_na]
```

Out[6]:

```
[('PlanetaryMassJpt', 2271),
 ('RadiusJpt', 810),
 ('PeriodDays', 99),
 ('SemiMajorAxisAU', 2178),
 ('Eccentricity', 2476),
 ('PeriastronDeg', 3256),
 ('LongitudeDeg', 3541),
 ('AscendingNodeDeg', 3538),
 ('InclinationDeg', 2919),
 ('SurfaceTempK', 2843),
 ('AgeGyr', 3582),
 ('DiscoveryMethod', 63),
 ('DiscoveryYear', 10),
 ('LastUpdated', 8),
 ('RightAscension', 10),
 ('Declination', 10),
 ('DistFromSunParsec', 1451),
 ('HostStarMassSlrMass', 168),
 ('HostStarRadiusSlrRad', 321),
 ('HostStarMetallicity', 1075),
 ('HostStarTempK', 129),
 ('HostStarAgeGyr', 3067)]
```

```
#удаление колонок, где пропуски преобладают
data=data.drop(['Eccentricity','PlanetaryMassJpt','AgeGyr','SemiMajorAxisA
U','LongitudeDeg','LongitudeDeg', 'PeriastronDeg','SurfaceTempK', 'Ascendi
ngNodeDeg', 'HostStarAgeGyr','InclinationDeg'], axis='columns')
data.dtypes
```

Out[10]:

```
PlanetIdentifier         object
TypeFlag                  int64
RadiusJpt               float64
PeriodDays              float64
DiscoveryMethod          object
DiscoveryYear           float64
LastUpdated              object
RightAscension           object
Declination              object
DistFromSunParsec       float64
```

```
HostStarMassSlrMass     float64
HostStarRadiusSlrRad    float64
HostStarMetallicity     float64
HostStarTempK           float64
ListsPlanetIsOn          object
dtype: object
```

Устранение пропусков в данных

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

Категориальные признаки

```
#заполнение наиболее частыми значениями
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data[['LastUpdated','RightAscension','Declination']] = imp.fit_transform(d
ata[['LastUpdated', 'RightAscension','Declination']])
#Введение отдельного значения категории для пропущенных значений
imp2 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_valu
e='no data')
data[['DiscoveryMethod']] = imp2.fit_transform(data[['DiscoveryMethod']])
data['DiscoveryMethod'].unique()
```

```
Out[21]:
array(['RV', 'transit', 'microlensing', 'no data', 'imaging', 'timing'],
      dtype=object)
```

Числовые признаки

```
#Заполнение средним значением data.loc[:, 'RadiusJpt']=data.loc[:,
'RadiusJpt'].fillna(data['RadiusJpt'].mean())

data.loc[:, 'DiscoveryYear']=data.loc[:,
'DiscoveryYear'].fillna(data['DiscoveryYear'].mean())
data.loc[:, 'HostStarMetallicity']=data.loc[:, 'HostStarMetallicity'].fill
na(data['HostStarMetallicity'].mean())
data.loc[:, 'HostStarRadiusSlrRad']=data.loc[:, 'HostStarRadiusSlrRad'].fi
llna(data['HostStarRadiusSlrRad'].mean())
```

Кодирование категориальных признаков

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
le.fit(data.PlanetIdentifier )
data['PlanetIdentifier']=le.transform(data.PlanetIdentifier )
```

:

```
#Count encoding
from category_encoders.count import CountEncoder as ce_CountEncoder
ce_CountEncoder1 = ce_CountEncoder()
data['DiscoveryMethod'] = ce_CountEncoder1.fit_transform(data['DiscoveryMe
thod'])

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

```
Out[23]:
array([ 692, 2712,   40,   63,   52,   25], dtype=int64)
```

```
le=LabelEncoder()
le.fit(data.LastUpdated )
data['LastUpdated']=le.transform(data.LastUpdated )

le=LabelEncoder() le.fit(data.RightAscension)
data['RightAscension']=le.transform(data.RightAscension)
le=LabelEncoder()
le.fit(data.Declination)
```

```
data['Declination']=le.transform(data.Declination)

le=LabelEncoder() le.fit(data.ListsPlanetIsOn)
data['ListsPlanetIsOn']=le.transform(data.ListsPlanetIsOn)
data.dtypes
```

Out[28]:

```
PlanetIdentifier          int32
TypeFlag                  int64
RadiusJpt               float64
PeriodDays              float64
DiscoveryMethod           int64
DiscoveryYear           float64
LastUpdated               int32
RightAscension            int32
Declination               int32
DistFromSunParsec       float64
HostStarMassSlrMass     float64
HostStarRadiusSlrRad    float64
HostStarMetallicity     float64
HostStarTempK           float64
ListsPlanetIsOn           int32
dtype: object
```

```
from sklearn.impute import KNNImputer
knnimputer = KNNImputer(
    n_neighbors=5,
    weights='distance',
    metric='nan_euclidean',
    add_indicator=False,
)
knnimpute_hdata_imputed_temp = knnimputer.fit_transform(data)
data = pd.DataFrame(knnimpute_hdata_imputed_temp, columns=data.columns)
data.head()
```

Out[30]:

	PlanetIdentifier	TypeFlag	RadiusJpt	PeriodDays	DiscoveryMethod	DiscoveryYear	LastUpdated	RightAscension	Declination	DistFromSunParsec	Hos
0	468.0	0.0	0.37119	39.845800	692.0	2016.000000	454.0	608.0	274.0	17.236000	0.88
1	469.0	0.0	0.37119	102.540000	692.0	2016.000000	454.0	608.0	274.0	17.236000	0.88
2	948.0	0.0	0.05400	0.176891	2712.0	2012.000000	239.0	972.0	549.0	176.802885	0.46
3	946.0	0.0	0.11400	4.194525	2712.0	2013.300504	440.0	972.0	549.0	175.005239	0.46
4	947.0	0.0	0.07100	6.356006	2712.0	2013.300504	440.0	972.0	549.0	175.032000	0.46

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

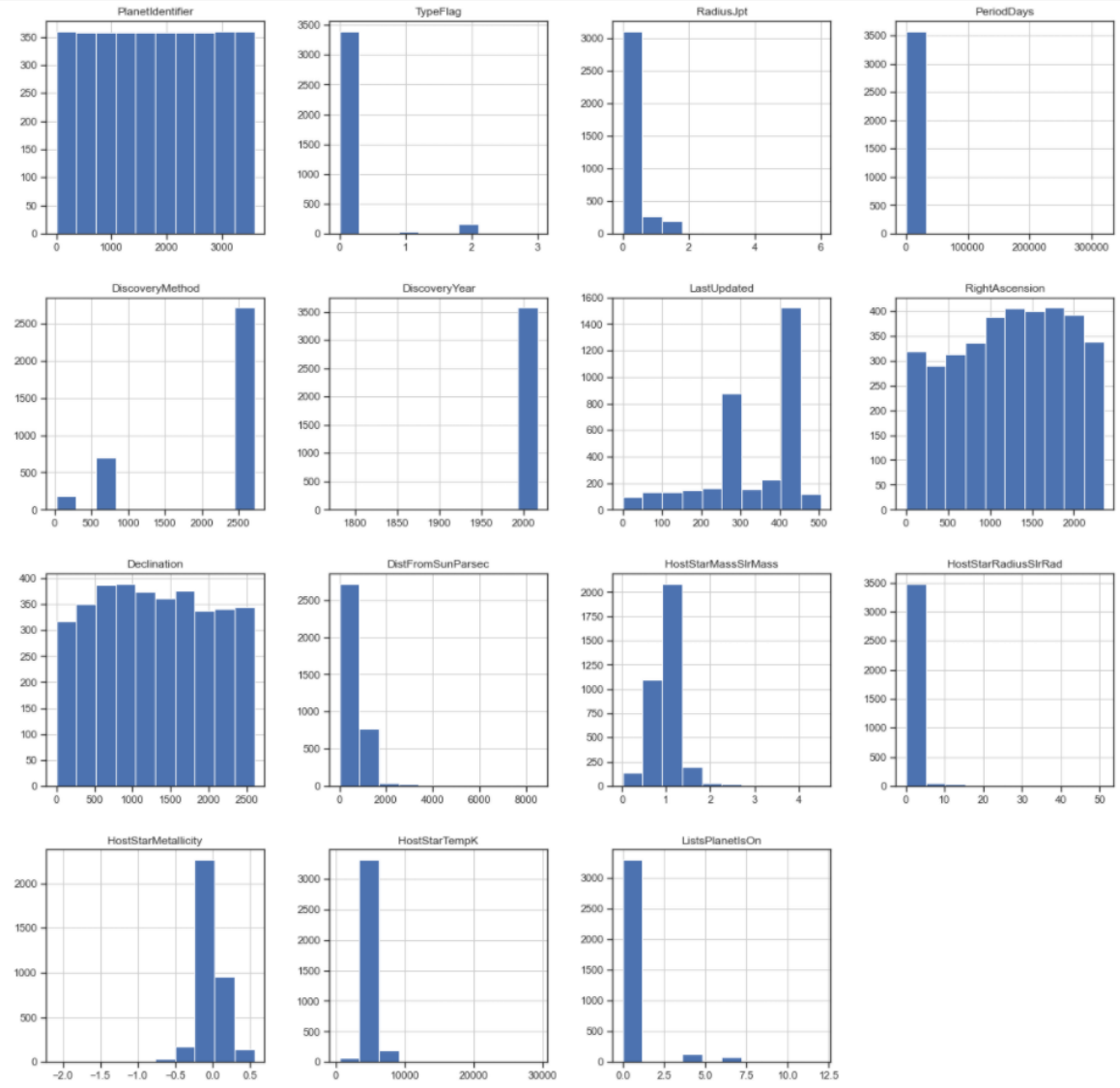
Out[31]:

```
PlanetIdentifier        0
TypeFlag                0
RadiusJpt               0
PeriodDays              0
DiscoveryMethod         0
DiscoveryYear           0
LastUpdated             0
RightAscension          0
Declination             0
DistFromSunParsec       0
HostStarMassSlrMass     0
HostStarRadiusSlrRad    0
```

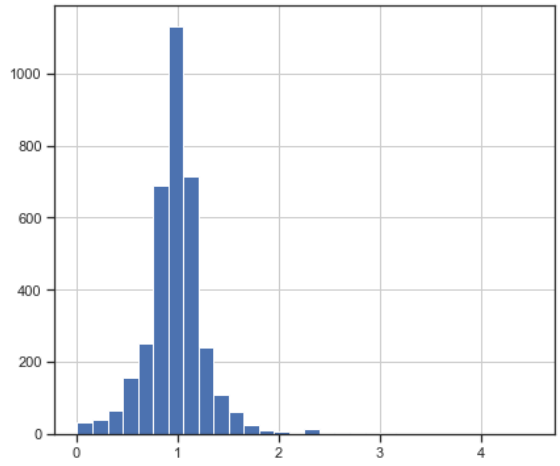
```
HostStarMetallicity     0
HostStarTempK           0
ListsPlanetIsOn         0
dtype: int64
```

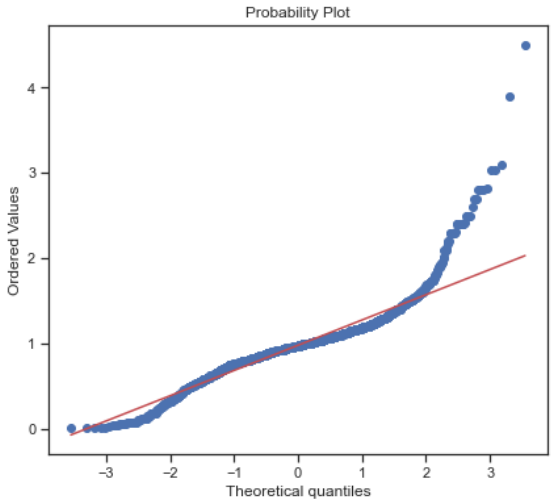
Нормализация числовых признаков

```
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()
```

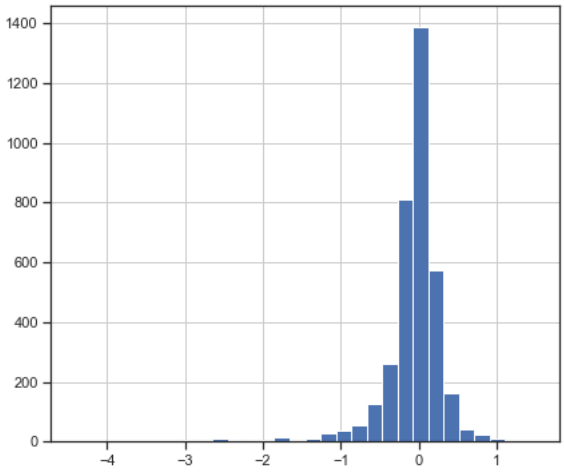


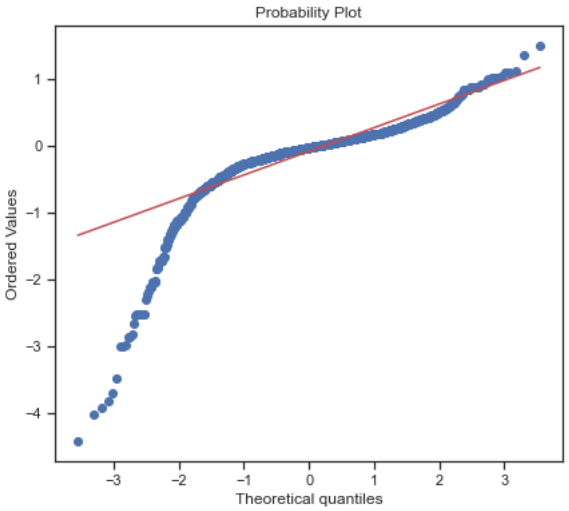

```
#Исходное распределение
diagnostic_plots(data, 'HostStarMassSlrMass')
```



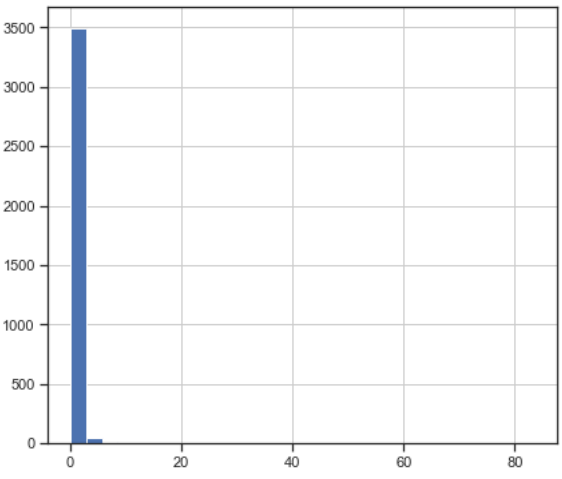


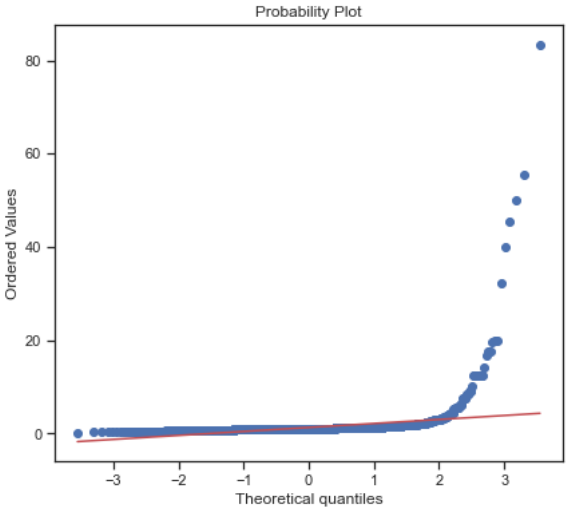

```
#Логарифмическое преобразование
data['HostStarMassSlrMass_log'] = np.log(data['HostStarMassSlrMass'])
diagnostic_plots(data, 'HostStarMassSlrMass_log')
```



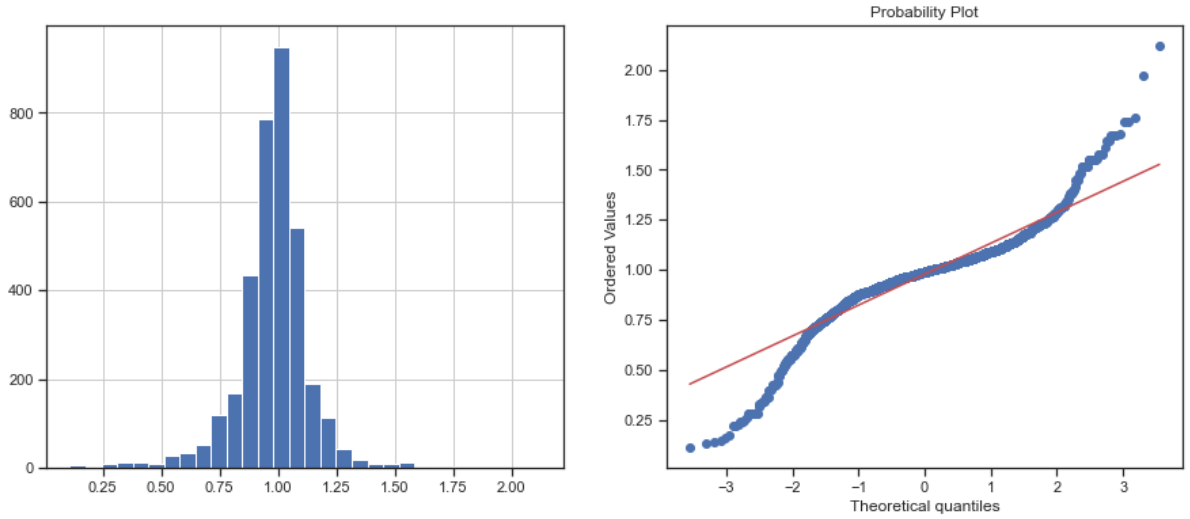



```
#Обратное преобразование
data['HostStarMassSlrMass_reciprocal'] = 1 / (data['HostStarMassSlrMass'])
diagnostic_plots(data, 'HostStarMassSlrMass_reciprocal')
```

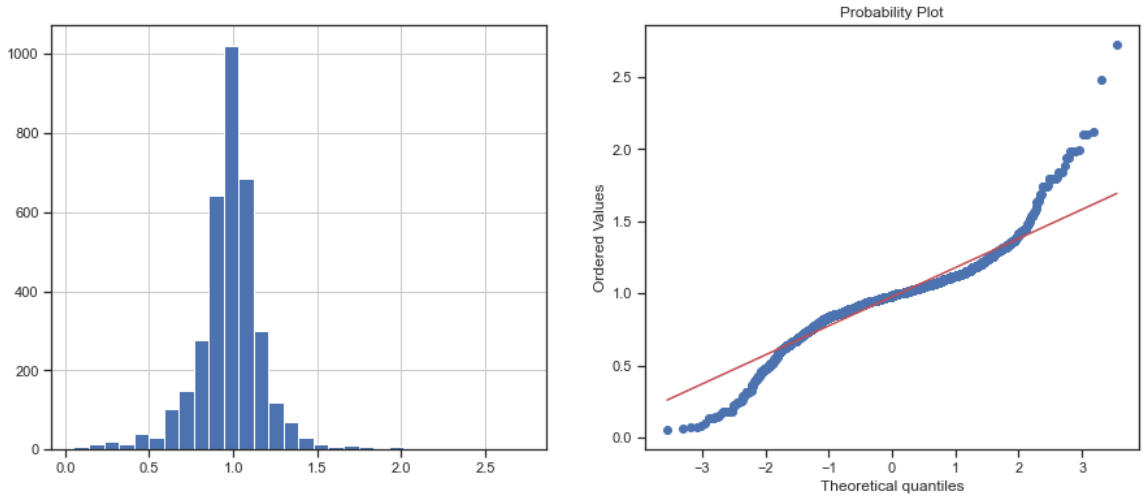




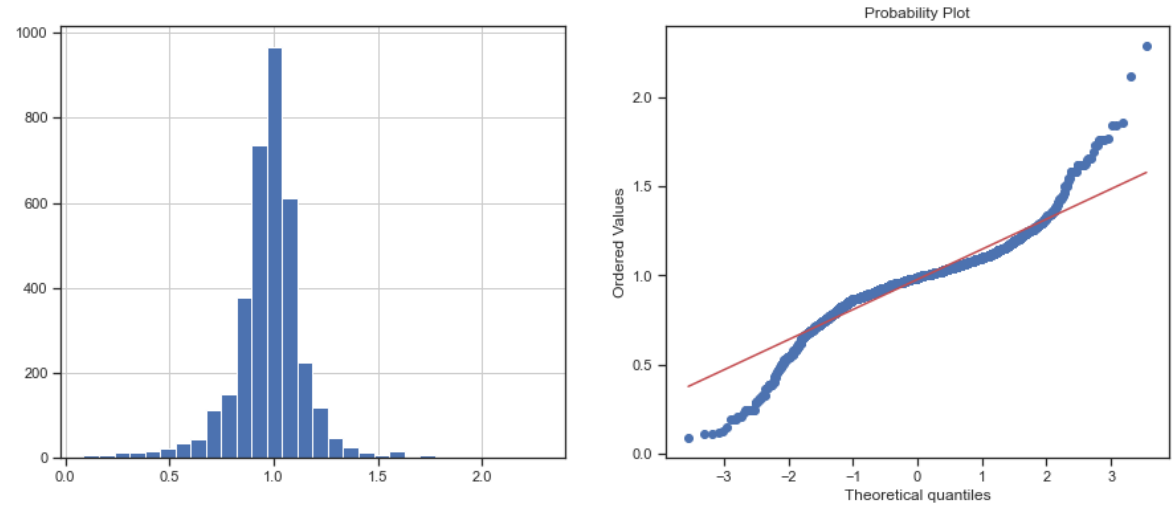
```
#Квадратный корень
data['HostStarMassSlrMass_sqr'] = data['HostStarMassSlrMass']**(1/2)
diagnostic_plots(data, 'HostStarMassSlrMass_sqr')
```




```
: #Возведение в степень
data['HostStarMassSlrMass_exp1'] = data['HostStarMassSlrMass']**(1/1.5)
diagnostic_plots(data, 'HostStarMassSlrMass_exp1')
```

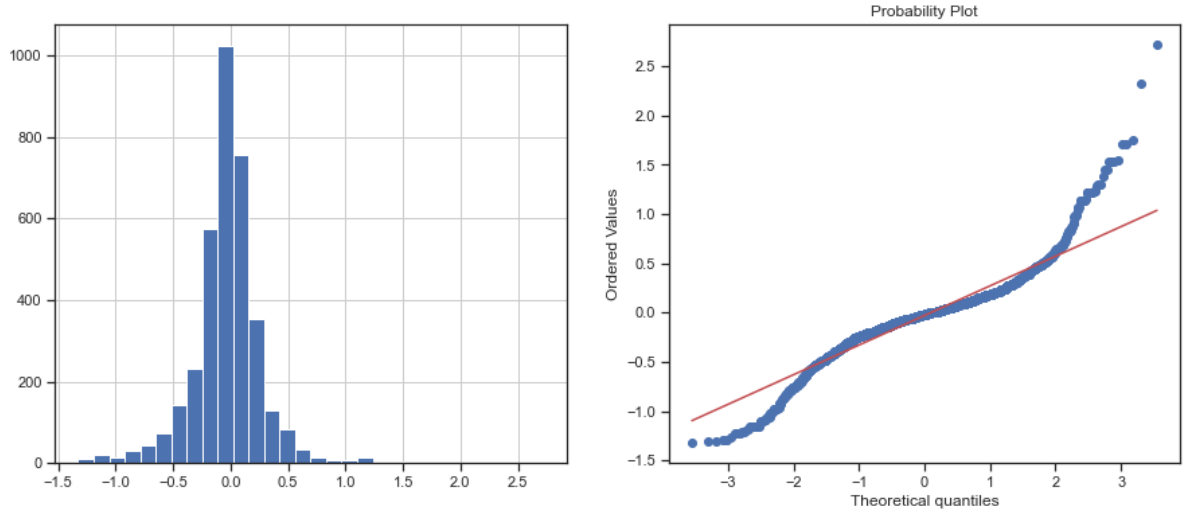



```
|: data['HostStarMassSlrMass_exp3'] = data['HostStarMassSlrMass']**(0.55)
diagnostic_plots(data, 'HostStarMassSlrMass_exp3')
```



```
#Преобразование Бокса-Кокса
data['HostStarMassSlrMass_boxcox'], param = stats.boxcox(data['HostStarMassSlrMass'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(data, 'HostStarMassSlrMass_boxcox')
```

```
Оптимальное значение λ = 0.7231760307645071
```




```
#Преобразование Йео-Джонсона
data['HostStarMassSlrMass_yeojohnson'], param = stats.yeojohnson(data['HostStarMassSlrMass'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(data, 'HostStarMassSlrMass_yeojohnson')
```

```
Оптимальное значение λ = 0.16838591624404547
```