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
Лю Ченхао

Москва — 2024 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.) Просьба не использовать датасет, на котором данная задача решалась в лекции.
2. Для выбранного датасета (датасетов) на основе материалов лекций решить следующие задачи:
    i. устранение пропусков в данных;
    ii. кодирование категориальных признаков;
    iii. нормализация числовых признаков.

Загрузка и первичный анализ данных

[1]:
```
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

[2]:
```
data = pd.read_csv(r'HR.csv')
```

[3]:
```
data.shape
```

[3]:
```
(1470, 10)
```

[4]:
```
data.isnull().sum()
```

[4]:
```
Age                  0
Attrition            7
BusinessTravel       9
DailyRate            0
DistanceFromHome    12
Education            5
Gender              14
HourlyRate           0
MaritalStatus        9
MonthlyIncome       15
dtype: int64
```

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

[5]:

	Age	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	Gender	HourlyRate	MaritalStatus	MonthlyIncome
0	41	Yes	Travel_Rarely	1102	1.0	2.0	Female	94	Single	NaN
1	49	No	Travel_Frequently	279	8.0	1.0	Male	61	Married	5130.0
2	37	Yes	Travel_Rarely	1373	2.0	2.0	Male	92	Single	2090.0
3	33	No	Travel_Frequently	1392	NaN	4.0	Female	56	Married	2909.0
4	27	No	Travel_Rarely	591	2.0	1.0	Male	40	Married	3468.0

Обработка пропусков в данных

[6]:
```python
# типы колонок
data.dtypes
```

[6]:
```
Age                  int64
Attrition           object
BusinessTravel      object
DailyRate            int64
DistanceFromHome   float64
Education          float64
Gender              object
HourlyRate           int64
MaritalStatus       object
MonthlyIncome      float64
dtype: object
```

[7]:
```python
data_clean = res = data.dropna(axis=1, how='any')
data_clean.shape
```

[7]:
```
(1470, 3)
```

[8]:
```python
data.isnull().sum()
```

[8]:
```
Age                  0
Attrition            7
BusinessTravel       9
DailyRate            0
DistanceFromHome    12
Education            5
Gender              14
HourlyRate           0
MaritalStatus        9
MonthlyIncome       15
dtype: int64
```

[9]:
```python
# Колонки с пропусками
cols_with_na = [c for c in data.columns if data[c].isnull().sum() > 0]
cols_with_na
```

[9]:
```
['Attrition',
 'BusinessTravel',
 'DistanceFromHome',
 'Education',
 'Gender',
 'MaritalStatus',
 'MonthlyIncome']
```

[10]:
```python
# Доля (процент) пропусков
[(c, data[c].isnull().mean()) for c in cols_with_na]
```

[10]:
```
[('Attrition', 0.004761904761904762),
 ('BusinessTravel', 0.006122448979591836),
 ('DistanceFromHome', 0.00816326530612245),
 ('Education', 0.003401360544217687),
 ('Gender', 0.009523809523809525),
 ('MaritalStatus', 0.006122448979591836),
 ('MonthlyIncome', 0.01020408163265306)]
```

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

Заполнение значений для одного признака

[11]:
```
# 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]]
```

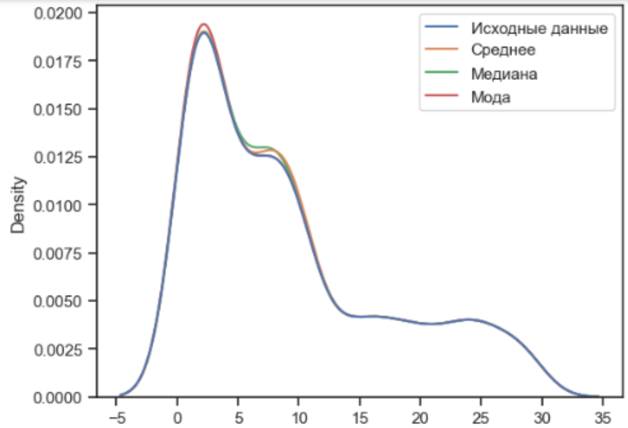
[12]:

```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
```

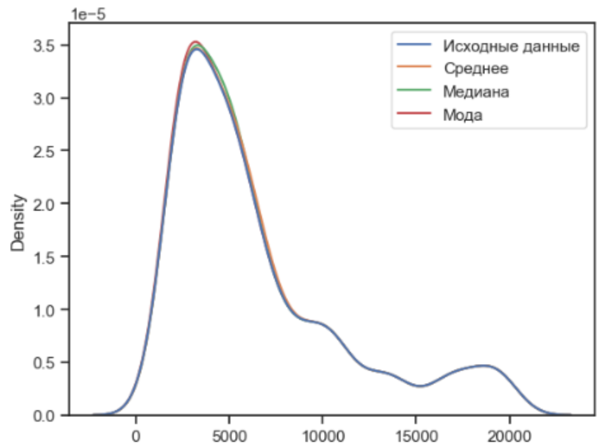
[13]:

```python
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)
```

```
[14]: research_impute_numeric_column(data, 'DistanceFromHome')
```




```
[15]: research_impute_numeric_column(data, 'MonthlyIncome')
```



Распределения одномодальные, поэтому можно использовать для импутации моду.

```
[16]: data_imp=data.copy(deep=True)
      LoanAmount_new, _,_ = impute_column(data_imp, 'DistanceFromHome', 'most_frequent')
```

```
[17]: data_imp['DistanceFromHome']=LoanAmount_new
```

```
[18]: data_imp.isnull().sum()
```

```
[18]: Age                  0
      Attrition            7
      BusinessTravel       9
      DailyRate            0
      DistanceFromHome     0
      Education            5
      Gender              14
      HourlyRate           0
      MaritalStatus        9
      MonthlyIncome       15
      dtype: int64
```

Для категориальных признаков

[19]:
```
hdata_cat_cols = ['Attrition', 'BusinessTravel', 'Gender','MaritalStatus']
hdata_cat_new = data[hdata_cat_cols].copy(deep=True)
```

[20]:
```

Attrition_cat_new_temp, _, _ = impute_column(hdata_cat_new, 'Attrition', 'most_frequent')
BusinessTravel_cat_new_temp, _, _ = impute_column(hdata_cat_new, 'BusinessTravel', 'most_frequent')
Gender_new_temp, _, _ = impute_column(hdata_cat_new, 'Gender', 'most_frequent')
MaritalStatus_new_temp, _, _ = impute_column(hdata_cat_new, 'MaritalStatus', 'most_frequent')
```

[21]:
```
hdata_cat_new['Attrition'] = Attrition_cat_new_temp
hdata_cat_new['BusinessTravel'] = BusinessTravel_cat_new_temp
hdata_cat_new['Gender'] = Gender_new_temp
hdata_cat_new['MaritalStatus'] = MaritalStatus_new_temp
```

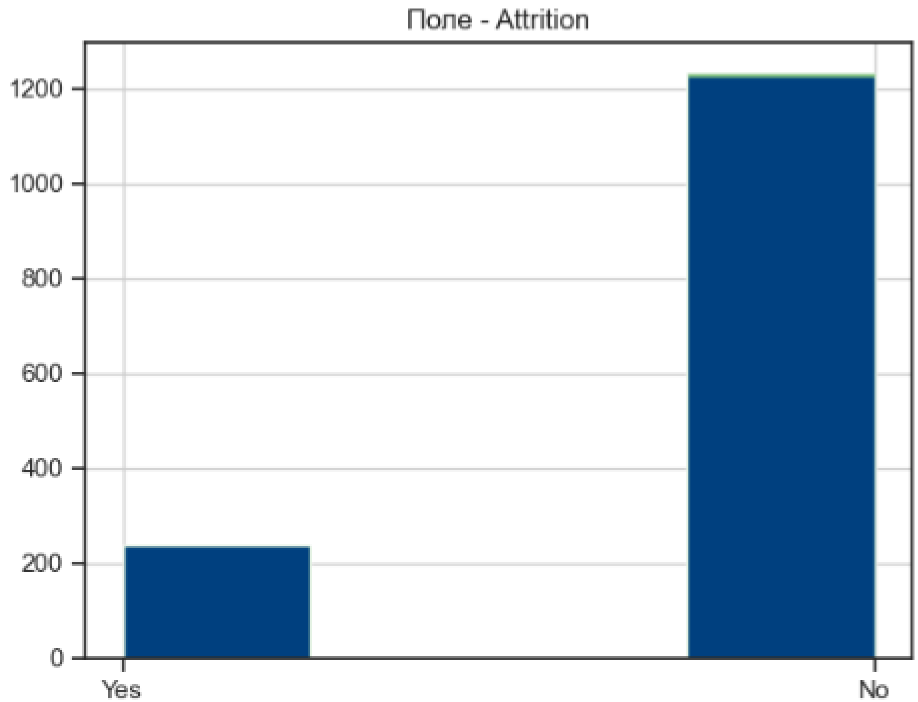
[22]:
```
data_imp_cat=data.copy()
data_imp_cat['Attrition'] = Attrition_cat_new_temp
data_imp_cat['BusinessTravel'] = BusinessTravel_cat_new_temp
data_imp_cat['Gender'] = Gender_new_temp
data_imp_cat['MaritalStatus'] = MaritalStatus_new_temp
```

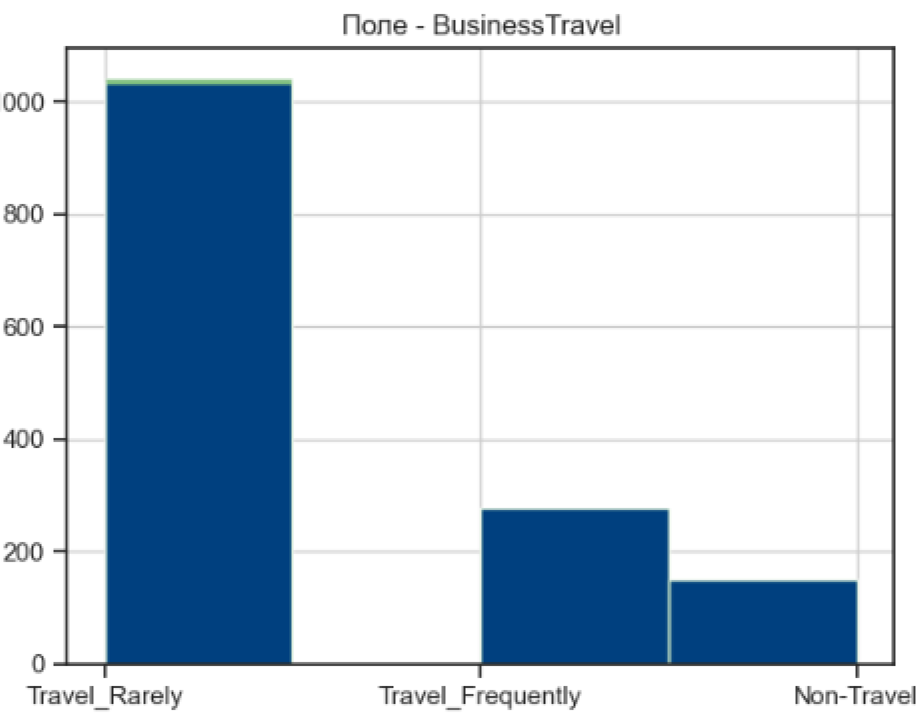
[23]:
```
data_imp_cat.isnull().sum()
```

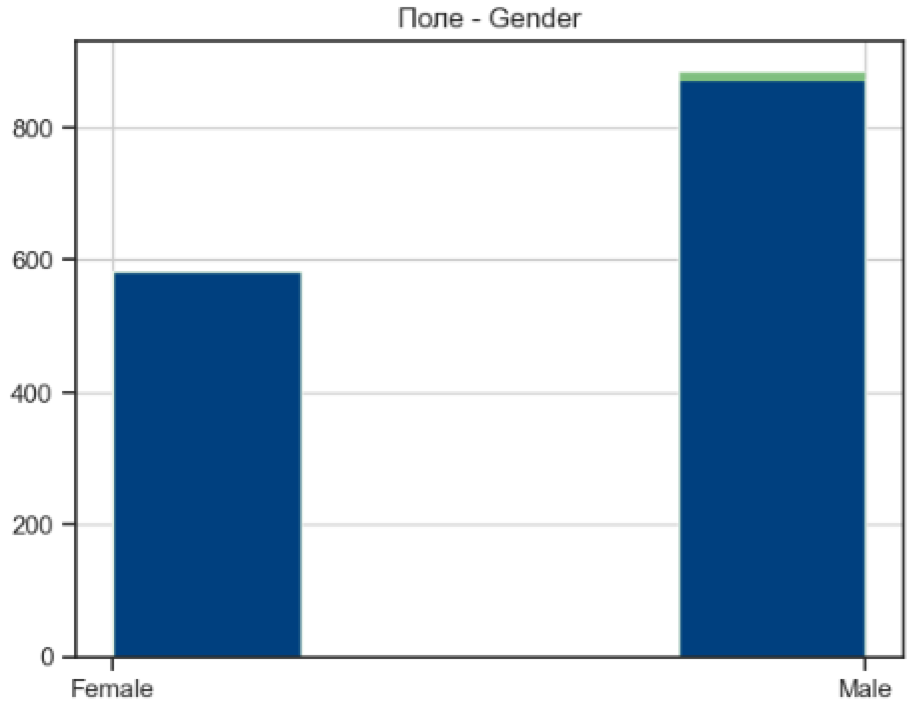
[23]:
```
Age                  0
Attrition            0
BusinessTravel       0
DailyRate            0
DistanceFromHome    12
Education            5
Gender               0
HourlyRate           0
MaritalStatus        0
MonthlyIncome       15
dtype: int64
```

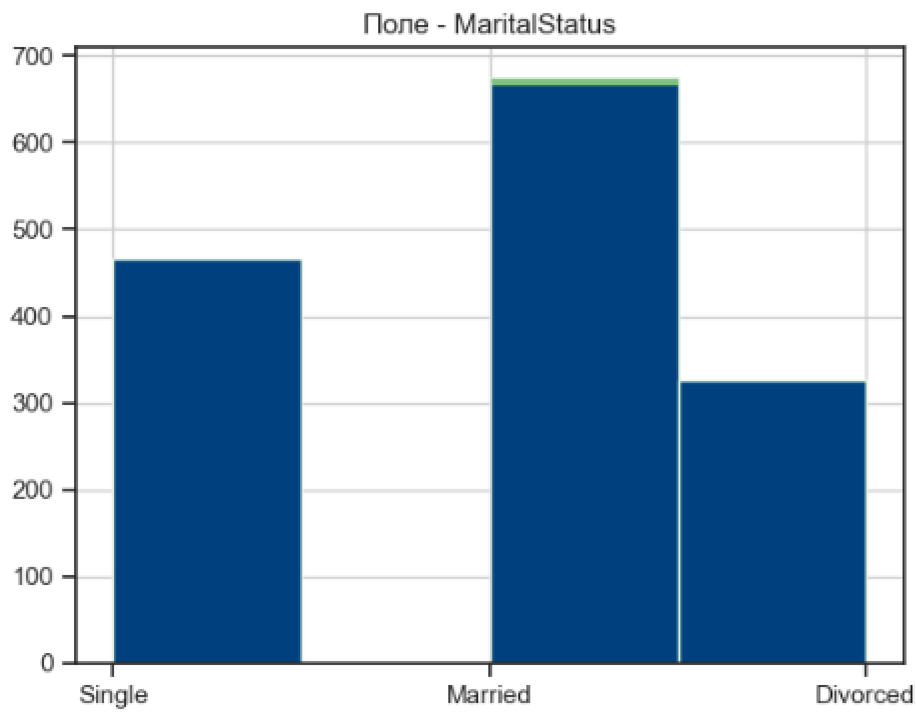
[24]:
```
def plot_hist_diff(old_ds, new_ds, cols):
    """

    """
    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()
```

[25]:
```
plot_hist_diff(data, hdata_cat_new, hdata_cat_cols)
```






Поле - Gender
0
200
400
600
800
Female
Male


Поле - MaritalStatus
0
100
200
300
400
500
600
700
Single
Married
Divorced

KNN

[26]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_knn=data_imp_cat.copy(deep=True) # устранены пропуски строковых признаков
data_knn['Attrition'] = le.fit_transform(data['Attrition'])
data_knn['BusinessTravel'] = le.fit_transform(data['BusinessTravel'])
data_knn['Gender'] = le.fit_transform(data['Gender'])
data_knn['MaritalStatus'] = le.fit_transform(data['MaritalStatus'])
```

[27]:
```
knnimpute_cols = ['Attrition', 'BusinessTravel','DistanceFromHome', 'Gender',]
```

[28]:
```
knnimpute_hdata = data_knn[knnimpute_cols].copy()
knnimpute_hdata.head()
```

[28]:

	Attrition	BusinessTravel	DistanceFromHome	Gender
0	1	2	1.0	0
1	0	1	8.0	1
2	1	2	2.0	1
3	0	1	NaN	0
4	0	2	2.0	1

[29]:
```
# Признаки с пропусками
knnimpute_hdata.isnull().sum()
```

[29]:
```
Attrition            0
BusinessTravel       0
DistanceFromHome    12
Gender               0
dtype: int64
```

[30]:
```
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()
```

[30]:

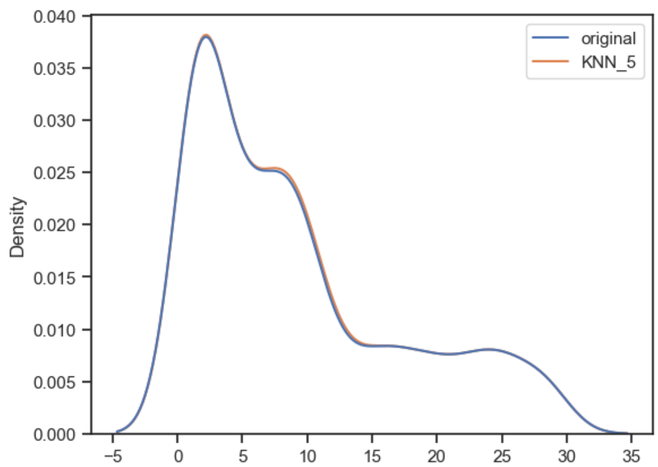
	Attrition	BusinessTravel	DistanceFromHome	Gender
0	1.0	2.0	1.0	0.0
1	0.0	1.0	8.0	1.0
2	1.0	2.0	2.0	1.0
3	0.0	1.0	2.8	0.0
4	0.0	2.0	2.0	1.0

[31]:
```
# Пропуски заполнены
knnimpute_hdata_imputed.isnull().sum()
```

[31]:
```
Attrition           0
BusinessTravel      0
DistanceFromHome    0
Gender              0
dtype: int64
```

[32]:
```
DistanceFromHome_df = pd.DataFrame({'original': knnimpute_hdata['DistanceFromHome'].values})
DistanceFromHome_df['KNN_5'] = knnimpute_hdata_imputed['DistanceFromHome']
sns.kdeplot(data=DistanceFromHome_df)
```

[32]: <Axes: ylabel='Density'>



кодирование категориальных признаков

[33]:
```
data_=data.copy(deep=True)
Age_new, _,_ = impute_column(data_, 'Age', 'most_frequent')
DailyRate_new, _,_ = impute_column(data_, 'DailyRate', 'most_frequent')
DistanceFromHome_new, _,_ = impute_column(data_, 'DistanceFromHome', 'most_frequent')
Education_new, _,_ = impute_column(data_, 'Education', 'most_frequent')
HourlyRate_new, _,_ = impute_column(data_, 'HourlyRate', 'most_frequent')
MonthlyIncome_new, _,_ = impute_column(data_, 'MonthlyIncome', 'most_frequent')

Attrition_cat_new_cat_new_temp, _,_ = impute_column(data_, 'Attrition', 'most_frequent')
BusinessTravel__cat_new_temp, _,_ = impute_column(data_, 'BusinessTravel', 'most_frequent')
Gender_cat_new_temp, _,_ = impute_column(data_, 'Gender', 'most_frequent')
MaritalStatus_cat_new_temp, _, _ = impute_column(data_, 'MaritalStatus', 'most_frequent')
```

[34]:
```
data_['Age'] = Age_new
data_['DailyRate']=DailyRate_new
data_['DistanceFromHome'] = DistanceFromHome_new
data_['Education'] = Education_new
data_['HourlyRate'] = HourlyRate_new
data_['MonthlyIncome'] = MonthlyIncome_new

data_['Attrition'] = Attrition_cat_new_temp
data_['BusinessTravel'] = BusinessTravel_cat_new_temp
data_['Gender'] = Gender_cat_new_temp
data_['MaritalStatus'] = MaritalStatus_cat_new_temp
```

[35]:
```
data_=data_.drop(['Age'], axis=1)
data_['Attrition'] = le.fit_transform(data_['Attrition'])
```

label encoding

[36]:
```
data_.head()
```

[36]:

	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	Gender	HourlyRate	MaritalStatus	MonthlyIncome
0	1	Travel_Rarely	1102	1.0	2.0	Female	94	Single	2342.0
1	0	Travel_Frequently	279	8.0	1.0	Male	61	Married	5130.0
2	1	Travel_Rarely	1373	2.0	2.0	Male	92	Single	2090.0
3	0	Travel_Frequently	1392	2.0	4.0	Female	56	Married	2909.0
4	0	Travel_Rarely	591	2.0	1.0	Male	40	Married	3468.0

[37]:
```
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['Attrition'] = le.fit_transform(data_Lable_enc['Attrition'])
data_Lable_enc['Education'] = le.fit_transform(data_Lable_enc['Education'])
data_Lable_enc['BusinessTravel'] = le.fit_transform(data_Lable_enc['BusinessTravel'])
```

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

[38]:
```
data_Lable_enc.head()
```

[38]:

	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	Gender	HourlyRate	MaritalStatus	MonthlyIncome
0	1	2	1102	1.0	1	0	94	Single	2342.0
1	0	1	279	8.0	0	1	61	Married	5130.0
2	1	2	1373	2.0	1	1	92	Single	2090.0
3	0	1	1392	2.0	3	0	56	Married	2909.0
4	0	2	591	2.0	0	1	40	Married	3468.0

one-hot encoding

[39]:
```
data_dumm=data_.copy(deep=True)
```

[40]:
```
# Добавление отдельной колонки, признака пустых значений
pd.get_dummies(data_dumm[['Gender']], dummy_na=True).head()
```

[40]:

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

[41]:
```
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
```

[42]:
```
data_dumm=encode_and_bind(data_dumm, 'Gender')
```

[43]:
```
data_dumm.head()
```

[43]:

	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	HourlyRate	MaritalStatus	MonthlyIncome	Gender_Female	Gender_Male
0	1	Travel_Rarely	1102	1.0	2.0	94	Single	2342.0	True	False
1	0	Travel_Frequently	279	8.0	1.0	61	Married	5130.0	False	True
2	1	Travel_Rarely	1373	2.0	2.0	92	Single	2090.0	False	True
3	0	Travel_Frequently	1392	2.0	4.0	56	Married	2909.0	True	False
4	0	Travel_Rarely	591	2.0	1.0	40	Married	3468.0	False	True

[44]:
```
data=data_.copy(deep=True)
```

[45]:
```
from category_encoders.one_hot import OneHotEncoder as ce_OneHotEncoder
```

[46]:
```
ce_OneHotEncoder1 = ce_OneHotEncoder()
data_OHE = ce_OneHotEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])])
```

[47]:
```
data_OHE
```

[47]:

	BusinessTravel_1	BusinessTravel_2	BusinessTravel_3	DailyRate	DistanceFromHome	Education	Gender_1	Gender_2	HourlyRate	MaritalStatus_1	Marit
0	1	0	0	1102	1.0	2.0	1	0	94	1	
1	0	1	0	279	8.0	1.0	0	1	61	0	
2	1	0	0	1373	2.0	2.0	0	1	92	1	
3	0	1	0	1392	2.0	4.0	1	0	56	0	
4	1	0	0	591	2.0	1.0	0	1	40	0	
...	...	...	...	...	...	...	...	...	...	...	
1465	0	1	0	884	23.0	2.0	0	1	41	0	
1466	1	0	0	613	6.0	1.0	0	1	42	0	
1467	1	0	0	155	4.0	3.0	0	1	87	0	
1468	0	1	0	1023	2.0	3.0	0	1	63	0	
1469	1	0	0	628	8.0	3.0	0	1	82	0	

1470 rows × 13 columns

Count (frequency) encoding

[48]:
```
from category_encoders.count import CountEncoder as ce_CountEncoder
```

[49]:
```
data_
```

[49]:

	Attrition	BusinessTravel	DailyRate	DistanceFromHome	Education	Gender	HourlyRate	MaritalStatus	MonthlyIncome
0	1	Travel_Rarely	1102	1.0	2.0	Female	94	Single	2342.0
1	0	Travel_Frequently	279	8.0	1.0	Male	61	Married	5130.0
2	1	Travel_Rarely	1373	2.0	2.0	Male	92	Single	2090.0
3	0	Travel_Frequently	1392	2.0	4.0	Female	56	Married	2909.0
4	0	Travel_Rarely	591	2.0	1.0	Male	40	Married	3468.0
...	...	...	...	...	...	...	...	...	...
1465	0	Travel_Frequently	884	23.0	2.0	Male	41	Married	2571.0
1466	0	Travel_Rarely	613	6.0	1.0	Male	42	Married	9991.0
1467	0	Travel_Rarely	155	4.0	3.0	Male	87	Married	6142.0
1468	0	Travel_Frequently	1023	2.0	3.0	Male	63	Married	5390.0
1469	0	Travel_Rarely	628	8.0	3.0	Male	82	Married	4404.0

1470 rows × 9 columns

[50]:
```
ce_CountEncoder1 = ce_CountEncoder()
data_COUNT_ENC = ce_CountEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])])
```

[51]:
```
data_COUNT_ENC
```

[51]:

	BusinessTravel	DailyRate	DistanceFromHome	Education	Gender	HourlyRate	MaritalStatus	MonthlyIncome
0	1043	1102	1.0	2.0	584	94	467	2342.0
1	277	279	8.0	1.0	886	61	676	5130.0
2	1043	1373	2.0	2.0	886	92	467	2090.0
3	277	1392	2.0	4.0	584	56	676	2909.0
4	1043	591	2.0	1.0	886	40	676	3468.0
...	...	...	...	...	...	...	...	...
1465	277	884	23.0	2.0	886	41	676	2571.0
1466	1043	613	6.0	1.0	886	42	676	9991.0
1467	1043	155	4.0	3.0	886	87	676	6142.0
1468	277	1023	2.0	3.0	886	63	676	5390.0
1469	1043	628	8.0	3.0	886	82	676	4404.0

1470 rows × 8 columns

[52]:
```
data_['MaritalStatus'].unique()
```

[52]:
```
array(['Single', 'Married', 'Divorced'], dtype=object)
```

[53]:
```
data_COUNT_ENC['MaritalStatus'].unique()
```

[53]:
```
array([467, 676, 327], dtype=int64)
```

Target (Mean) encoding

[54]:
```
# На самом деле этот метод реализует Mean encoding
from category_encoders.target_encoder import TargetEncoder as ce_TargetEncoder
```

[55]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
#data_6=data.copy(deep=True)
#data_6['Loan_Status'] = le.fit_transform(data_6['Loan_Status'])
```

[56]:
```
ce_TargetEncoder1 = ce_TargetEncoder()
data_MEAN_ENC = ce_TargetEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])], data_['Attrition'])
```

[57]:
```
data_MEAN_ENC
```

[57]:

	BusinessTravel	DailyRate	DistanceFromHome	Education	Gender	HourlyRate	MaritalStatus	MonthlyIncome
0	0.148610	1102	1.0	2.0	0.14726	94	0.254818	2342.0
1	0.249097	279	8.0	1.0	0.16930	61	0.124260	5130.0
2	0.148610	1373	2.0	2.0	0.16930	92	0.254818	2090.0
3	0.249097	1392	2.0	4.0	0.14726	56	0.124260	2909.0
4	0.148610	591	2.0	1.0	0.16930	40	0.124260	3468.0
...	...	...	...	...	...	...	...	...
1465	0.249097	884	23.0	2.0	0.16930	41	0.124260	2571.0
1466	0.148610	613	6.0	1.0	0.16930	42	0.124260	9991.0
1467	0.148610	155	4.0	3.0	0.16930	87	0.124260	6142.0
1468	0.249097	1023	2.0	3.0	0.16930	63	0.124260	5390.0
1469	0.148610	628	8.0	3.0	0.16930	82	0.124260	4404.0

1470 rows × 8 columns

[58]:
```
data_['MaritalStatus'].unique()
```

[58]:
```
array(['Single', 'Married', 'Divorced'], dtype=object)
```

[59]:
```
data_MEAN_ENC['MaritalStatus'].unique()
```

[59]:
```
array([0.25481799, 0.12426036, 0.10091743])
```

[60]:
```
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['Attrition']) / data_filter.shape[0]
            print(s, '-' , prob)
```

[61]:
```
check_mean_encoding('MaritalStatus')
```

```
Single - 0.25481798715203424
Married - 0.1242603550295858
Divorced - 0.10091743119266056
```

Weight of evidence (WoE) encoding

[62]:
```
from category_encoders.woe import WOEEncoder as ce_WOEEncoder
```

[63]:
```
ce_WOEEncoder1 = ce_WOEEncoder()
data_WOE_ENC = ce_WOEEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])], data_['Attrition'])
```

[64]:
```
data_WOE_ENC
```

[64]:

	BusinessTravel	DailyRate	DistanceFromHome	Education	Gender	HourlyRate	MaritalStatus	MonthlyIncome
0	-0.092876	1102	1.0	2.0	-0.099333	94	0.579785	2342.0
1	0.553526	279	8.0	1.0	0.062057	61	-0.295178	5130.0
2	-0.092876	1373	2.0	2.0	0.062057	92	0.579785	2090.0
3	0.553526	1392	2.0	4.0	-0.099333	56	-0.295178	2909.0
4	-0.092876	591	2.0	1.0	0.062057	40	-0.295178	3468.0
...	...	...	...	...	...	...	...	...
1465	0.553526	884	23.0	2.0	0.062057	41	-0.295178	2571.0
1466	-0.092876	613	6.0	1.0	0.062057	42	-0.295178	9991.0
1467	-0.092876	155	4.0	3.0	0.062057	87	-0.295178	6142.0
1468	0.553526	1023	2.0	3.0	0.062057	63	-0.295178	5390.0
1469	-0.092876	628	8.0	3.0	0.062057	82	-0.295178	4404.0

1470 rows × 8 columns

[65]:
```
# Проверка для поля "Пол"
data_['MaritalStatus'].unique()
```

[65]:
```
array(['Single', 'Married', 'Divorced'], dtype=object)
```

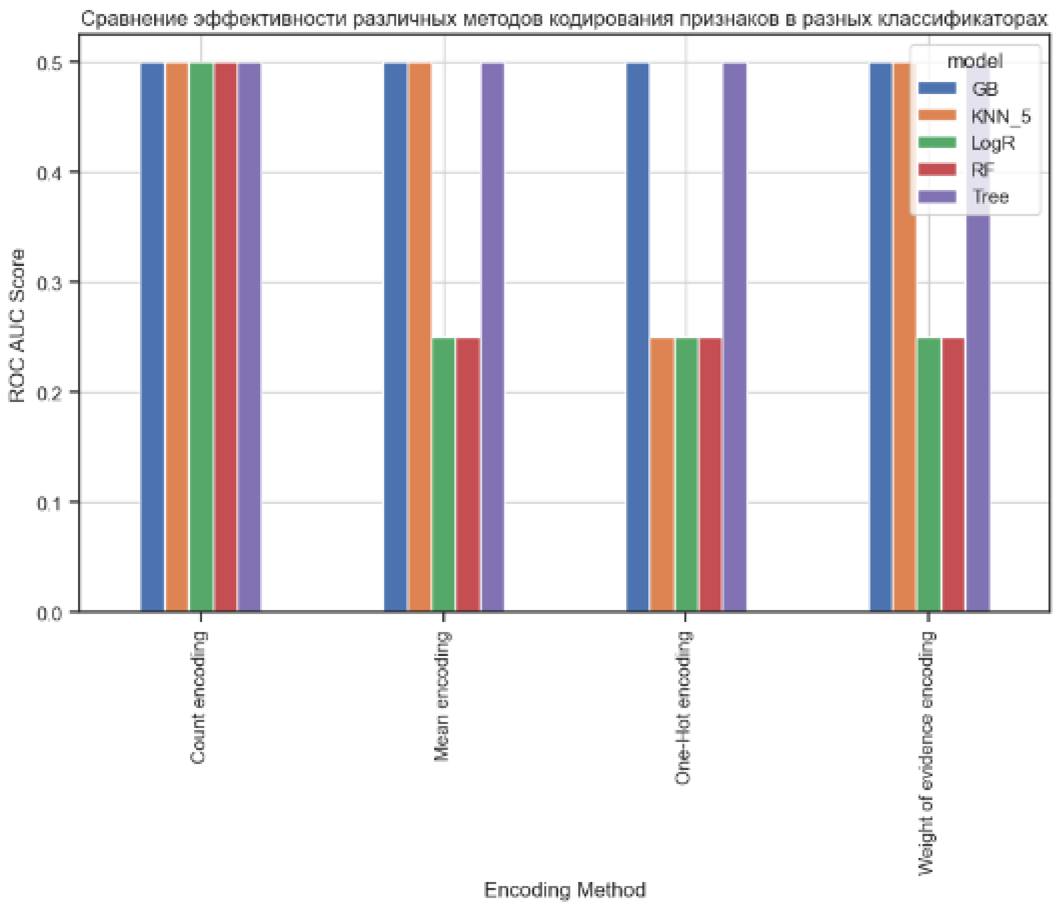
[66]:
```

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

[66]:
```
array([ 0.57978478, -0.29517818, -0.51324987])
```

```python
class MetricLogger:
    def __init__(self):
        self.df = pd.DataFrame(columns=['encoding', 'model', 'score'])

    def add(self, encoding, model, score):
        new_row = pd.DataFrame([[encoding, model, score]], columns=self.df.columns)
        if not new_row.empty:
            self.df = pd.concat([self.df, new_row], ignore_index=True)

    def plot(self, title, figsize=(10, 6)):
        plot_data = self.df.pivot_table(index='encoding', columns='model', values='score', aggfunc='mean')
        plot_data.plot(kind='bar', figsize=figsize)
        plt.title(title)
        plt.xlabel('Encoding Method')
        plt.ylabel('ROC AUC Score')
        plt.grid(True)
        plt.show()

# 测试不同编码方法和模型
def test_models(clas_models_dict, X_data_dict, y):
    logger = MetricLogger()
    for encoding, X in X_data_dict.items():
        # 检查和确保X和y的长度一致
        if len(X) != len(y):
            raise ValueError(f"Number of samples in {encoding} ({len(X)}) does not match the number of labels ({len(y)})")

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
        for model_name, model in clas_models_dict.items():
            model.fit(X_train, y_train)
            pred = model.predict_proba(X_test)
            roc_auc = roc_auc_score(y_test, pred[:, 1])
            logger.add(encoding, model_name, roc_auc)
    return logger

# 实例数据和编码
data_ = pd.DataFrame({
    'Gender': ['Male', 'Female', 'Male', 'Female', 'Female', 'Male', 'Male', 'Female', 'Male', 'Female'],
    'MaritalStatus': ['Single', 'Married', 'Single', 'Divorced', 'Married', 'Divorced', 'Single', 'Married', 'Divorced', 'Single'],
    'Attrition': [1, 0, 0, 1, 1, 0, 0, 1, 1, 0]
})

# 特征编码
from category_encoders.one_hot import OneHotEncoder as ce_OneHotEncoder
from category_encoders.count import CountEncoder as ce_CountEncoder
from category_encoders.target_encoder import TargetEncoder as ce_TargetEncoder
from category_encoders.woe import WOEEncoder as ce_WOEEncoder

# One-Hot Encoding
ce_OneHotEncoder1 = ce_OneHotEncoder()
data_OHE = ce_OneHotEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])])

# Count Encoding
ce_CountEncoder1 = ce_CountEncoder()
data_COUNT_ENC = ce_CountEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])])
```

```python
# One-Hot Encoding
ce_OneHotEncoder1 = ce_OneHotEncoder()
data_OHE = ce_OneHotEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])])

# Count Encoding
ce_CountEncoder1 = ce_CountEncoder()
data_COUNT_ENC = ce_CountEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])])

# Mean Encoding (Target Encoding)
ce_TargetEncoder1 = ce_TargetEncoder()
data_MEAN_ENC = ce_TargetEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])], data_['Attrition'])

# WoE Encoding
ce_WOEEncoder1 = ce_WOEEncoder()
data_WOE_ENC = ce_WOEEncoder1.fit_transform(data_[data_.columns.difference(['Attrition'])], data_['Attrition'])

# 标签数据
y = data_['Attrition']

# 将不同编码方法的数据集加载进来
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,
              }

# 定义分类器字典
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)
}

# 运行测试和绘图
logger = test_models(clas_models_dict, X_data_dict, y)
print(logger.df)
logger.plot('Сравнение эффективности различных методов кодирования признаков в разных классификаторах')
```

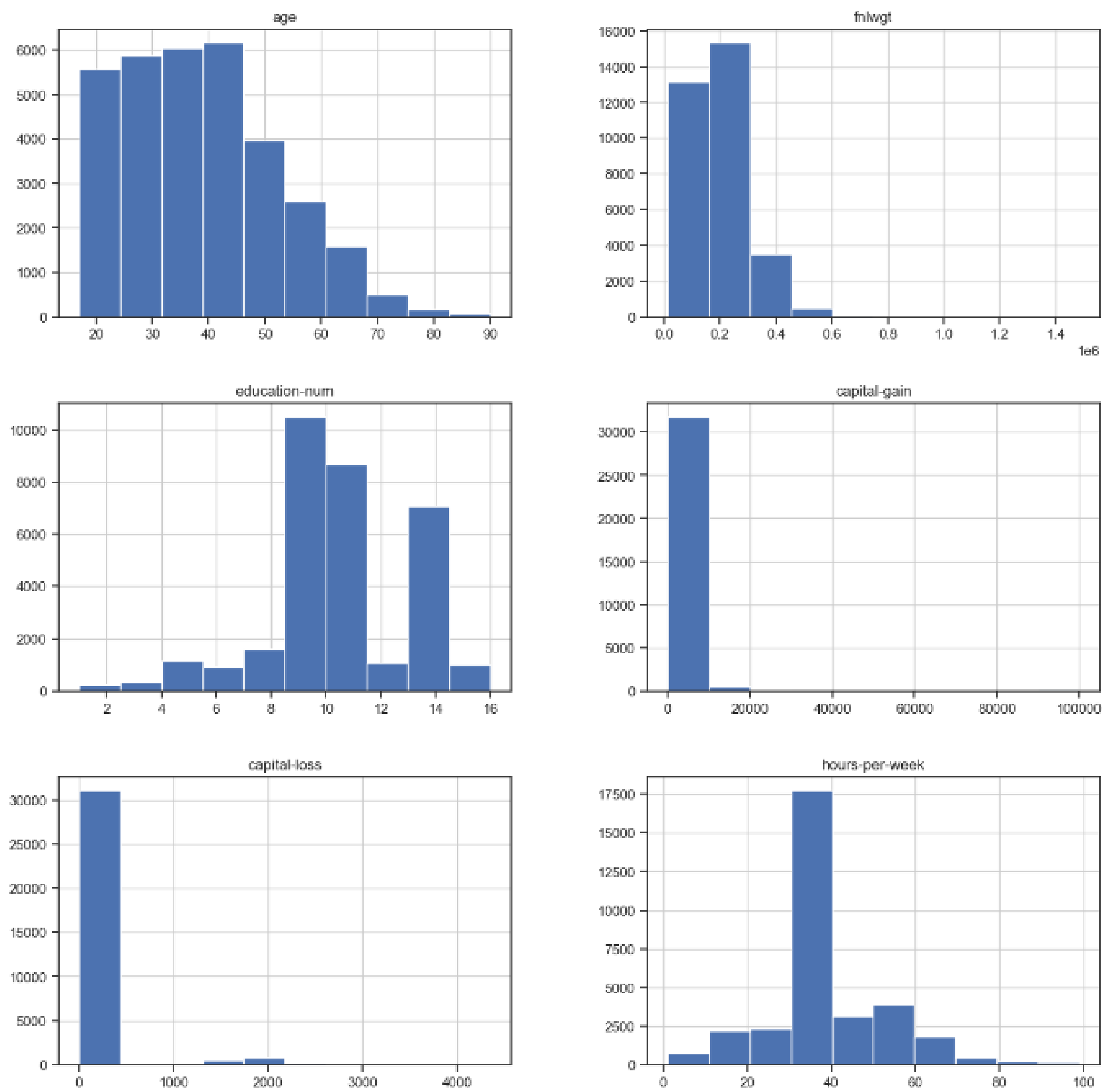
```
                      encoding   model  score
0              One-Hot encoding   LogR   0.25
1              One-Hot encoding  KNN_5   0.25
2              One-Hot encoding   Tree   0.50
3              One-Hot encoding     GB   0.50
4              One-Hot encoding     RF   0.25
5                Count encoding   LogR   0.50
6                Count encoding  KNN_5   0.50
7                Count encoding   Tree   0.50
8                Count encoding     GB   0.50
9                Count encoding     RF   0.50
10                Mean encoding   LogR   0.25
11                Mean encoding  KNN_5   0.50
12                Mean encoding   Tree   0.50
13                Mean encoding     GB   0.50
14                Mean encoding     RF   0.25
15  Weight of evidence encoding   LogR   0.25
16  Weight of evidence encoding  KNN_5   0.50
17  Weight of evidence encoding   Tree   0.50
18  Weight of evidence encoding     GB   0.50
19  Weight of evidence encoding     RF   0.25
```



нормализация числовых признаков

[68]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
```

[69]:
```python
data = pd.read_csv(r"adult.csv")
data.hist(figsize=(15, 15))
plt.show()
```



[70]:
```
data.head()
```

[70]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

[71]:
```
data.shape
```

[71]:
```
(32561, 15)
```

[72]:
```
data = data.dropna(subset=['age'], axis=0)
```

[73]:
```
data = data[data['age'] != 0]
```

[74]:
```
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()
```

[75]:
```
diagnostic_plots(data, 'age')
```

[70]:
```
data.head()
```

[70]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

[71]:
```
data.shape
```

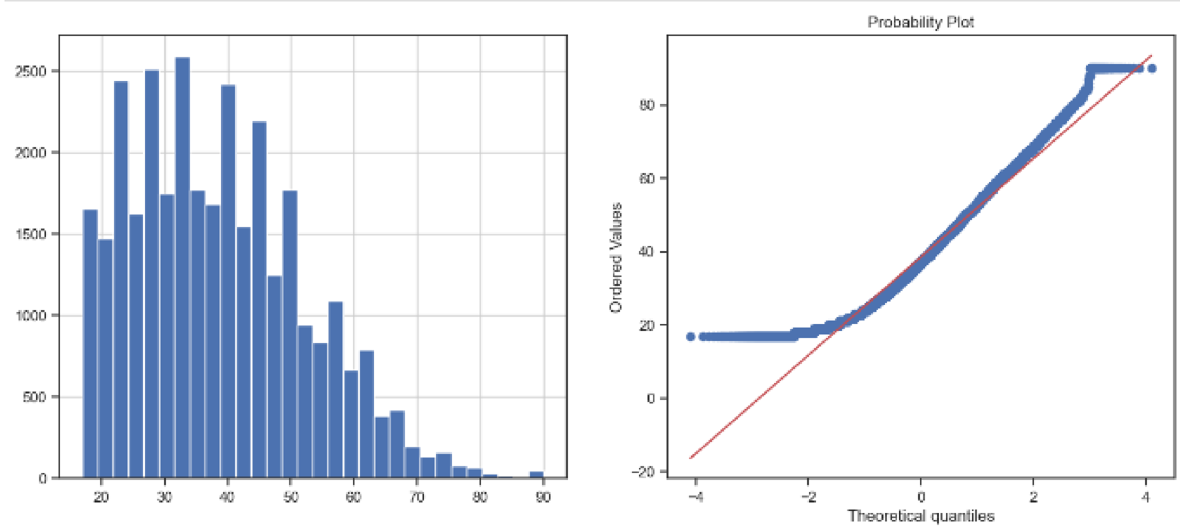
[71]:
```
(32561, 15)
```

[72]:
```
data = data.dropna(subset=['age'], axis=0)
```

[73]:
```
data = data[data['age'] != 0]
```

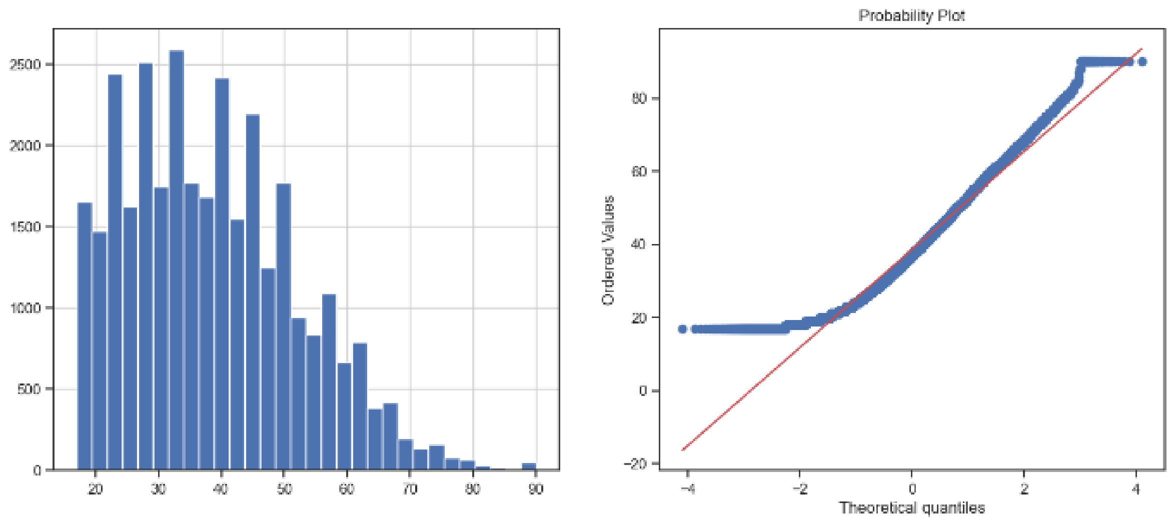
[74]:
```
def diagnostic_plots(df, variable):
    plt.figure(figsize=(15,6))
    # гистограмма
    plt.subplot(1, 2, 1)
    df[variable].hist(bins=30)
    ## Q-Q plot
    plt.subplot(1, 2, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.show()
```

[75]:
```
diagnostic_plots(data, 'age')
```



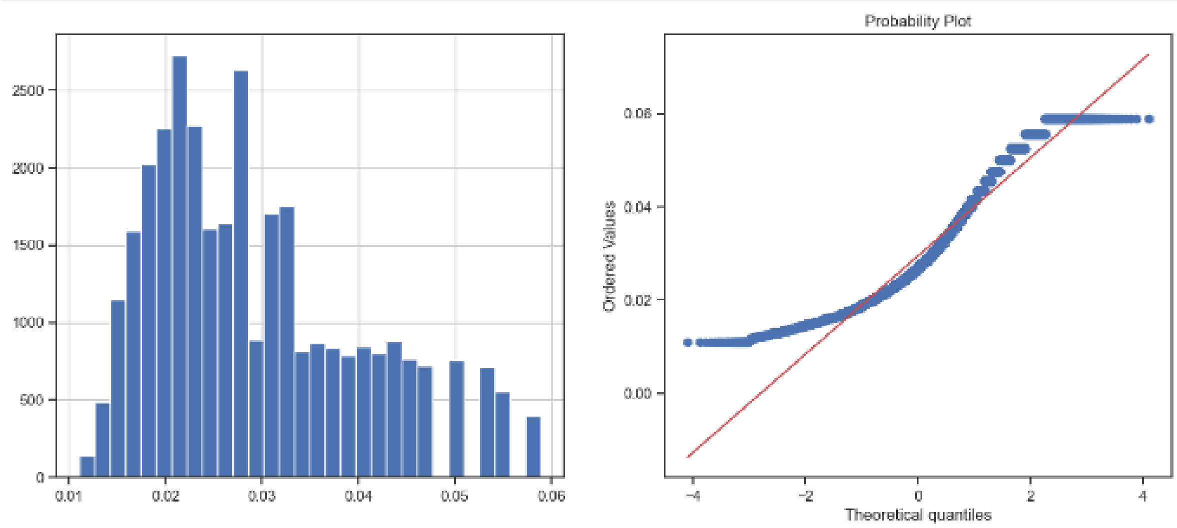
Логарифмическое преобразование

```
[76]: data['Perceptions of corruption log'] = np.log(data['age'])
      diagnostic_plots(data, 'age')
```



Обратное преобразование

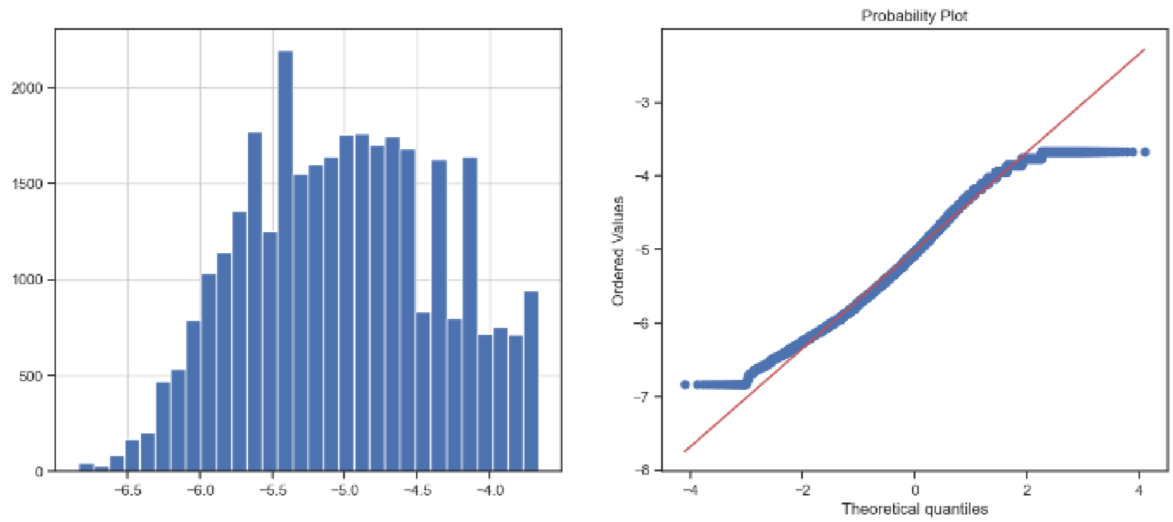
```
[77]: data['age'] = 1 / (data['age'])
      diagnostic_plots(data, 'age')
```



Преобразование Бокса-Кокса

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

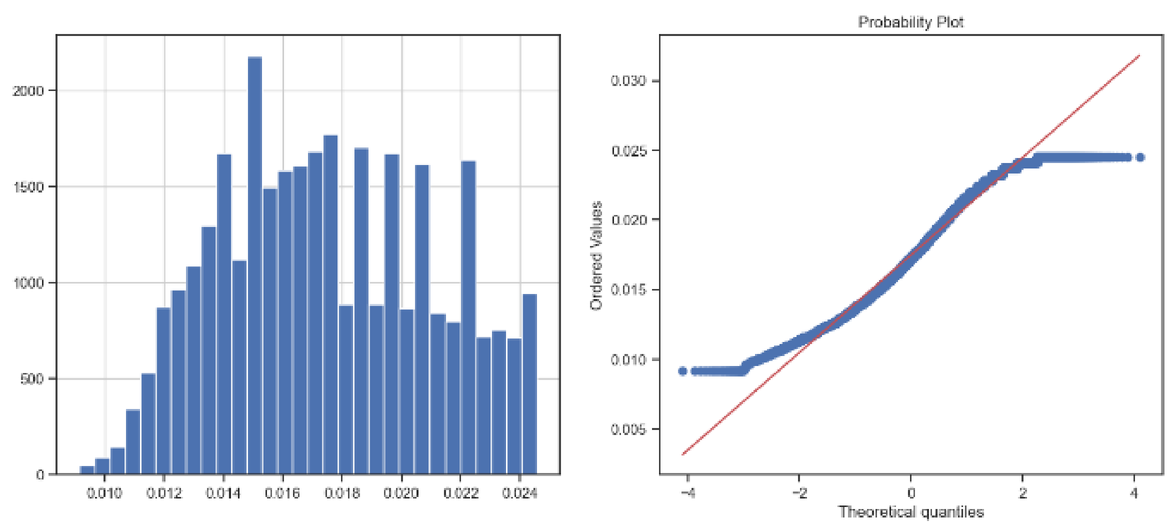
```
Оптимальное значение λ = -0.17459141591543673
```



Преобразование Йео-Джонсона

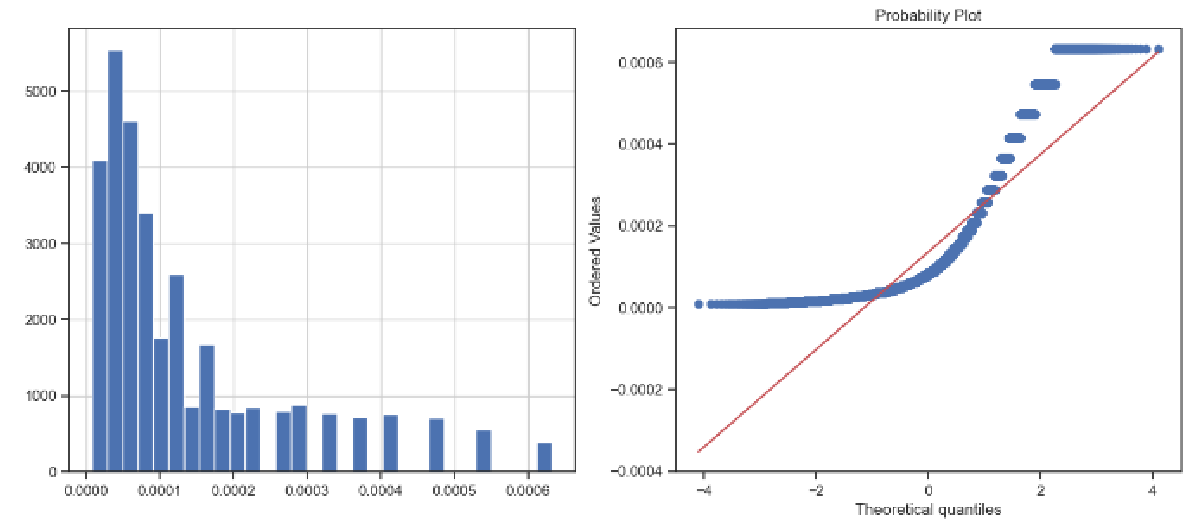
```
[79]: # Необходимо преобразовать данные к действительному типу

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

```
Оптимальное значение λ = -35.32360374319266
```



Возведение в степень

```
[80]: data['age exp'] = data['age'] ** (2.6)
      diagnostic_plots(data, 'age exp')
```



Список литературы

[1] Гапанюк Ю. Е. Лабораторная работа «Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных» .Access mode:https://github.com/ugapanyuk/courses_current/wiki/COURSE_MMO_SPRING_2024
[2] Team The IPython Development. IPython 7.3.0 Documentation //Read the Docs. — Access mode: https://ipython.readthedocs.io/en/stable/
[3] Waskom M. seaborn 0.9.0 documentation // PyData. . Access mode: https://seaborn.pydata.org/
[4] pandas 0.24.1 documentation [Electronic resource] // PyData.. — Access mode: http://pandas.pydata.org/pandas-docs/stable/
[5] Kaggle.. — Access mode: https://www.kaggle.com/ s