Лабораторная работа №2

Обработка признаков. Часть 1

Выполнил: Борисочкин М. И. ИУ5-21М

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.

Описание исходного набора данных

В качестве набора данных был взят Vehicle Sales Data, содержащий данные о продажах автомобилей и состоящий из следующих полей:

- year. Год производства автомобиля
- make. Брэнд автомобиля
- model. Модель автомобиля
- trim. Дополнение к модели автомобиля
- body. Тип двигателя
- transmission. Тип трансмиссии
- vin. ВИН номер
- state. Штат, где двигатель зарегистрирован
- condition. Состояние двигателя на момент продажи
- odometer. Значение на одометре машины перед продажей (пробег автомобиля)
- color. Цвет автомобиля
- interior. Цвет интерьера автомобиля
- seller. Продавец автомобиля
- mmr. Manheim Market Report. Может показывать рыночную цену автомобиля
- sellingprice. Цена, за которую продали автомобиль
- saledate. Дата и время, когда продали автомобиль

Импорт библиотек

In [1]:
```
from sklearn.impute import SimpleImputer, KNNImputer, MissingIndicator
from category_encoders.count import CountEncoder
from category_encoders.one_hot import OneHotEncoder

import numpy as np
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_style("whitegrid")
```

Загрузка датасета

In [2]:
```
initial_data = pd.read_csv("data/car_prices.csv")
```

In [3]:
```
# Размер датасета
initial_data.shape
```

Out[3]:
```
(558837, 16)
```

Изначальный датасет слишком большой. Сделаем подвыборку на 25 000 записей

In [4]:
```
# Создание нового датасета поменьше
data = initial_data.sample(25000, random_state=8)

data.head()
```

Out[4]:

	year	make	model	trim	body	transmission	vin	state
363218	2008	Mercury	Milan	Premier	Sedan	automatic	3mehm021x8r636447	ny
3460	2007	Ford	Fusion	SE	Sedan	automatic	3fahp07z07r258408	ca
508202	2014	Kia	Rio	LX	sedan	automatic	knadm4a38e6340309	nv
472373	2013	Ford	Escape	SEL	suv	automatic	1fmcu0hxxdud23355	il
273509	2013	Mercedes-Benz	C-Class	C250 Sport	Sedan	automatic	wddgf4hb1da816103	ca

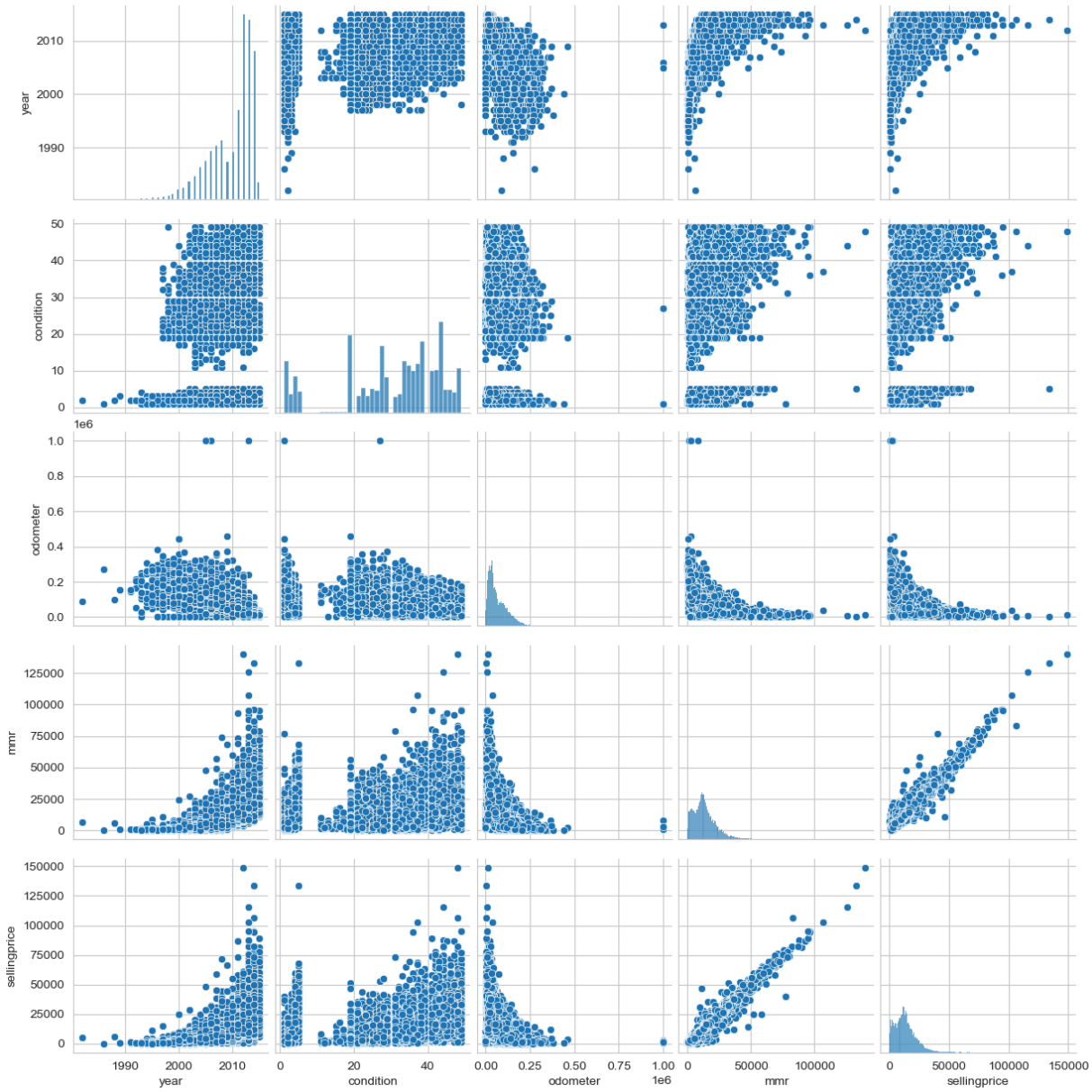
In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 25000 entries, 363218 to 210765
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   year          25000 non-null  int64  
 1   make          24541 non-null  object 
 2   model         24538 non-null  object 
 3   trim          24528 non-null  object 
 4   body          24404 non-null  object 
 5   transmission  22121 non-null  object 
 6   vin           25000 non-null  object 
 7   state         25000 non-null  object 
 8   condition     24473 non-null  float64
 9   odometer      24994 non-null  float64
 10  color         24965 non-null  object 
 11  interior      24965 non-null  object 
 12  seller        25000 non-null  object 
 13  mmr           25000 non-null  float64
 14  sellingprice  25000 non-null  float64
 15  saledate      25000 non-null  object 
dtypes: float64(4), int64(1), object(11)
memory usage: 3.2+ MB
```

In [6]:
```
# Нулевые значения в наборе данных
data.isnull().sum()
```

Out[6]:
```
year              0
make            459
model           462
trim            472
body            596
transmission   2879
vin               0
state             0
condition       527
odometer          6
color            35
interior         35
seller            0
mmr               0
sellingprice      0
saledate          0
dtype: int64
```

In [7]:
```
sns.pairplot(data)
plt.show()
```



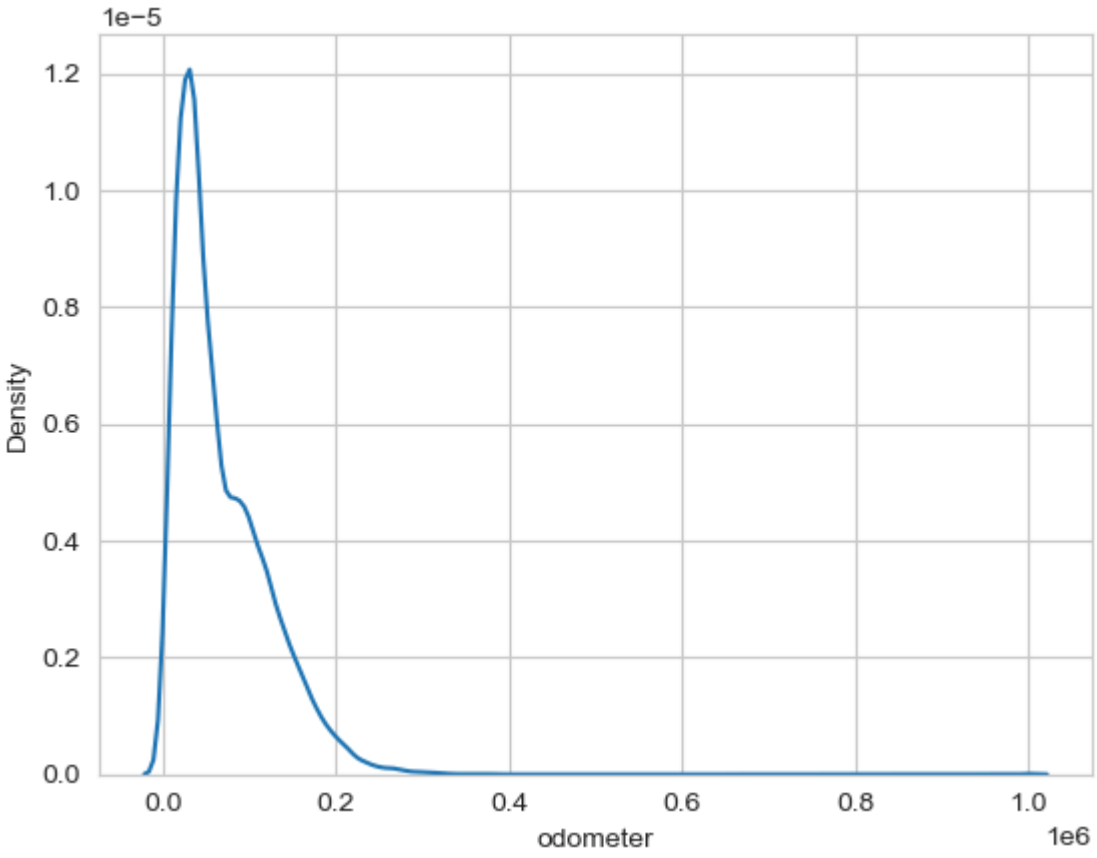
Устранение пропусков

Первым делом разберёмся с пропусками в числовых признаках, т. к. оных с пропусками гораздо меньше, чем категориальных.

In [8]:
```
# Количество пропусков в числовых колонках в %
numbers_missing_cols = ["condition", "odometer"]

for column in numbers_missing_cols:
    print(f" Пропусков в колонке {column}: {data[column].isnull().sum()/data[column
```

```
Пропусков в колонке condition: 2.108 %
Пропусков в колонке odometer: 0.024 %
```

In [9]:
```
sns.kdeplot(data=data, x="odometer")
plt.show()
```

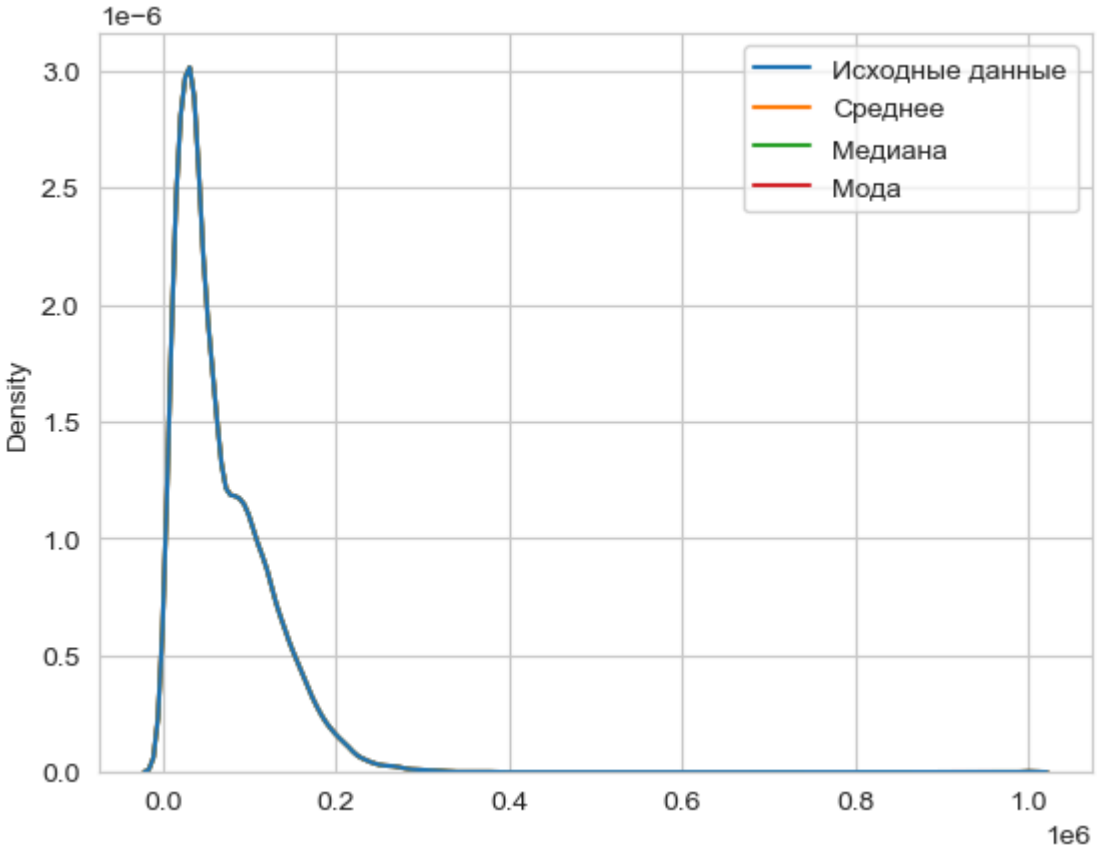


In [10]:
```python
def impute_column(dataset, column, strategy_param, fill_value_param=None):
    """
    Заполнение пропусков в одном признаке
    """
    temp_data = dataset[[column]].values
    size = temp_data.shape[0]

    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 [11]:
```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
            else:
                temp_data, _, _ = impute_column(dataset, num_column, strategy)
            new_df[col_name] = temp_data

    sns.kdeplot(data=new_df)
```

In [12]:
```
# Различные стратегии заполнения для признака odometer
research_impute_numeric_column(data, "odometer")
```

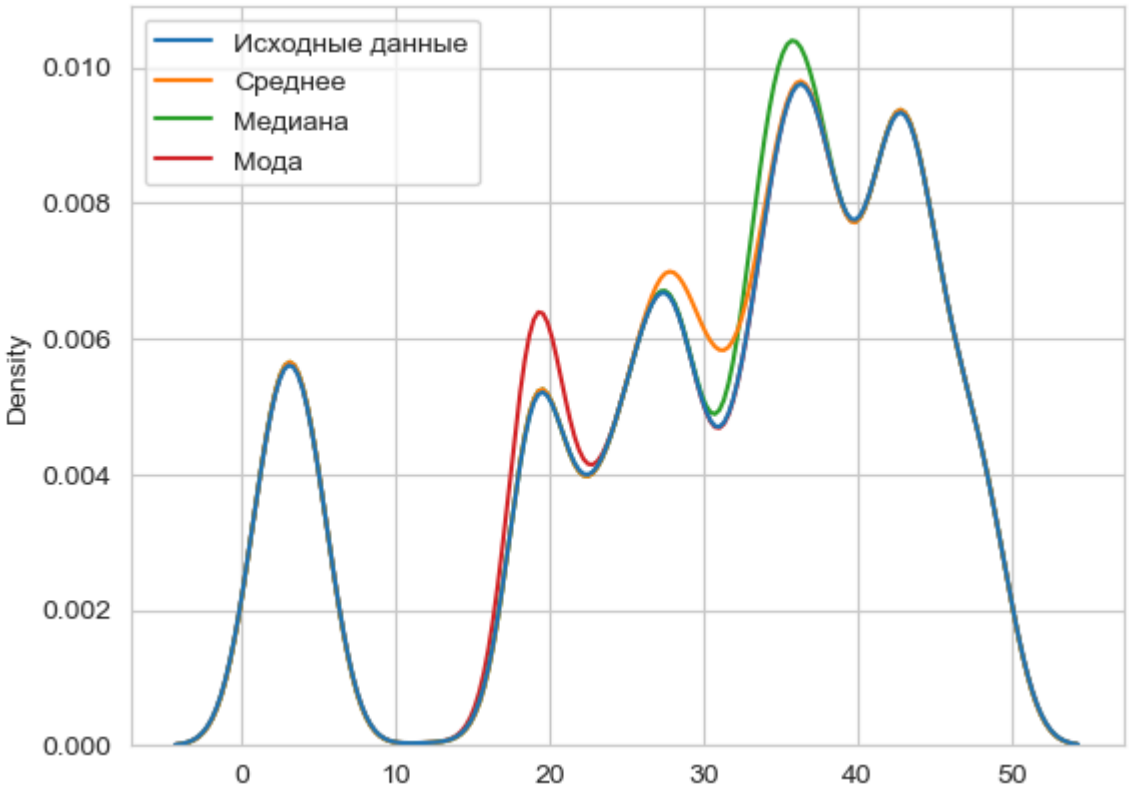


Стратегия заполнения признака odometer не имеет значения, так как пропусков слишком мало. Заполним пропущенные значения модой

In [13]:
```
# Заполнение пустых значений в поле odometer
data_filled = data.copy().reset_index(drop=True)
data_filled["odometer"], _, _ = impute_column(data_filled, "odometer", strategy_par

data_filled["odometer"].isnull().sum()
```

Out[13]:
```
0
```

In [14]:
```
# Различные стратегии заполнения для признака condition
research_impute_numeric_column(data, "condition")
```



Признак condition предлагаю заполнить с помощью KNN.

In [15]:
```
# Список колонок, которые будут участвовать в KNNInpute
knnimpute_cols = numbers_missing_cols + ["year", "mmr"]
knnimpute_cols
```

Out[15]:
```
['condition', 'odometer', 'year', 'mmr']
```

In [16]:
```
# До вставки значений
knnimpute_df = data_filled[knnimpute_cols].copy()
knnimpute_df.isnull().sum()
```

Out[16]:
```
condition    527
odometer       0
year           0
mmr            0
dtype: int64
```

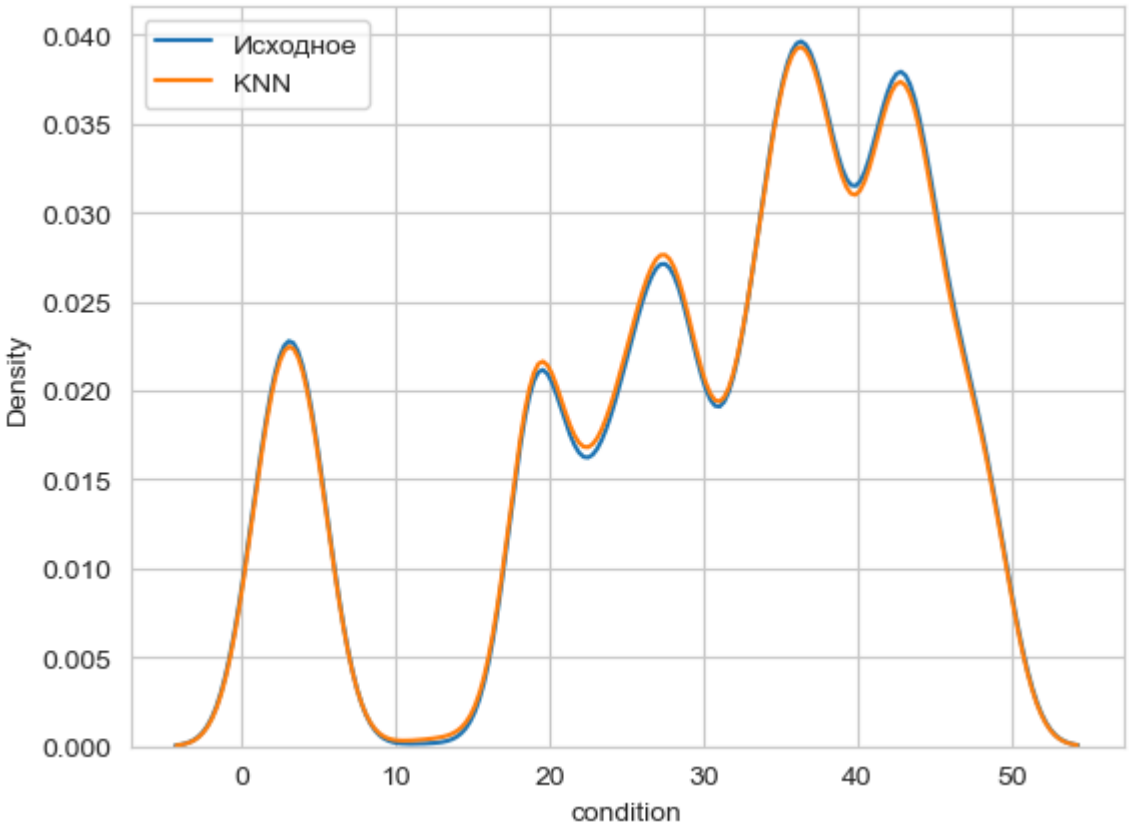
In [17]:
```
# Создание импютера и вставка пропущенных значений
knnimputer = KNNImputer(
    n_neighbors=5,
    weights='distance',
    metric='nan_euclidean',
    add_indicator=False,
)
```

```
knnimpute_df_imputed_temp = knnimputer.fit_transform(knnimpute_df)
knnimpute_df_imputed = pd.DataFrame(knnimpute_df_imputed_temp, columns=knnimpute_df
```

In [18]:
```
# После вставки значений
knnimpute_df_imputed.isnull().sum()
```

Out[18]:
```
condition    0
odometer     0
year         0
mmr          0
dtype: int64
```

In [19]:
```
# KDE до заполнения пропусков и после
sns.kdeplot(data=data_filled, x="condition", label="Исходное")
sns.kdeplot(data=knnimpute_df_imputed, x="condition", label="KNN")
plt.legend()
plt.show()
```



In [20]:
```
# Применение импьютации
data_filled["condition"] = knnimpute_df_imputed["condition"].astype("float64")
data_filled["condition"].isnull().sum()
```

Out[20]:
```
0
```

Теперь будем разбираться с категориальными признаками

In [21]:
```
# Количество пропусков в категориальных колонках в %
cat_missing_cols = ["make", "model", "trim", "body", "transmission", "color", "inte
```

```
for column in cat_missing_cols:
    print(f" Пропусков в колонке {column}: {data[column].isnull().sum()/data[column
```

```
Пропусков в колонке make: 1.836 %
Пропусков в колонке model: 1.848 %
Пропусков в колонке trim: 1.888 %
Пропусков в колонке body: 2.384 %
Пропусков в колонке transmission: 11.516 %
Пропусков в колонке color: 0.140 %
Пропусков в колонке interior: 0.140 %
```

In [22]:
```
# Количество уникальных значений в колонках
for column in cat_missing_cols:
    print (f"Количество уникальных значений в колонке {column}: {len((data_filled[c
```

```
Количество уникальных значений в колонке make: 73
Количество уникальных значений в колонке model: 658
Количество уникальных значений в колонке trim: 989
Количество уникальных значений в колонке body: 62
Количество уникальных значений в колонке transmission: 2
Количество уникальных значений в колонке color: 19
Количество уникальных значений в колонке interior: 17
```

Предлагаю удалить строки с пропущенными значениями во всех признаках, кроме transmition. Также проверим насколько в процентах уменьшилась входная выборка

In [23]:
```
data_filled_cat = data_filled.dropna(subset=["make", "model", "trim", "body", "colo

for column in cat_missing_cols:
    print(f" Пропусков в колонке {column}: {data_filled_cat[column].isnull().sum()/
```

```
Пропусков в колонке make: 0.000 %
Пропусков в колонке model: 0.000 %
Пропусков в колонке trim: 0.000 %
Пропусков в колонке body: 0.000 %
Пропусков в колонке transmission: 11.120 %
Пропусков в колонке color: 0.000 %
Пропусков в колонке interior: 0.000 %
```

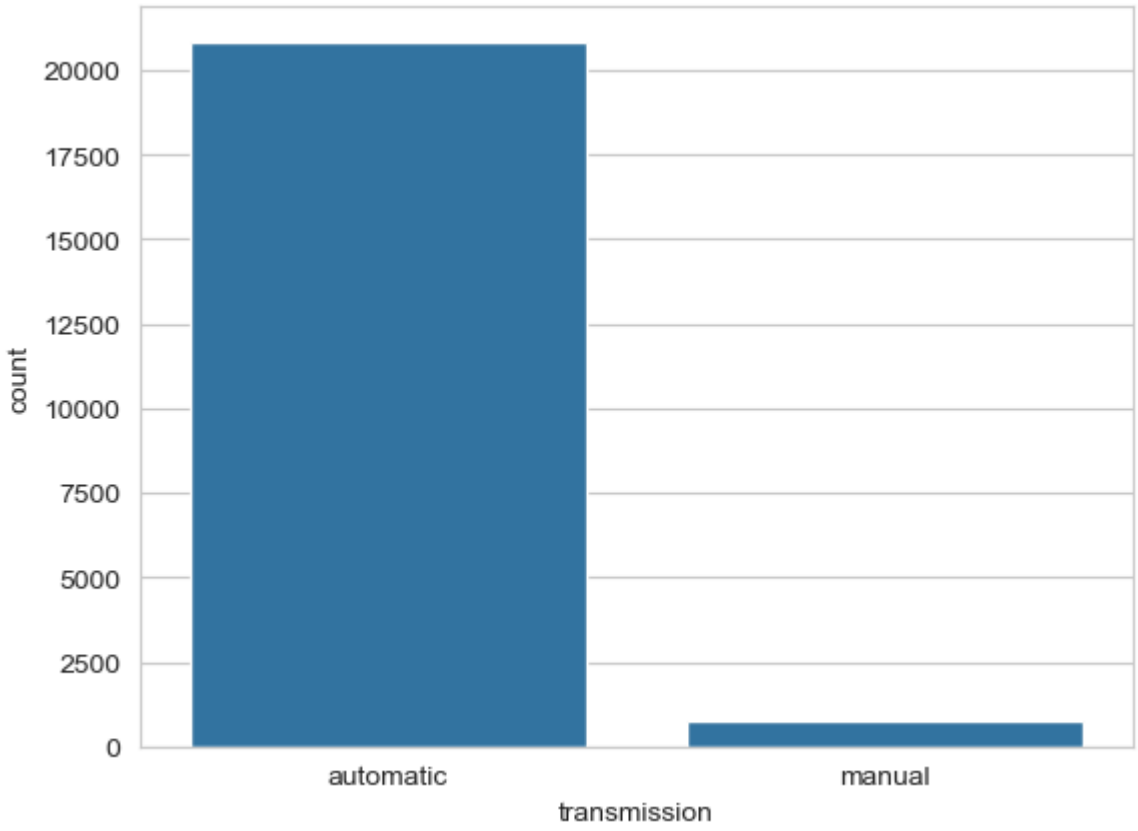
In [24]:
```
# Уменьшение размера выборки в процентах
(data.shape[0] - data_filled_cat.shape[0])/data.shape[0]*100
```

Out[24]: 2.528

Исходный набор данных уменьшился на 2,5 %, поэтому удаление пустых значений можно считать допустимым.

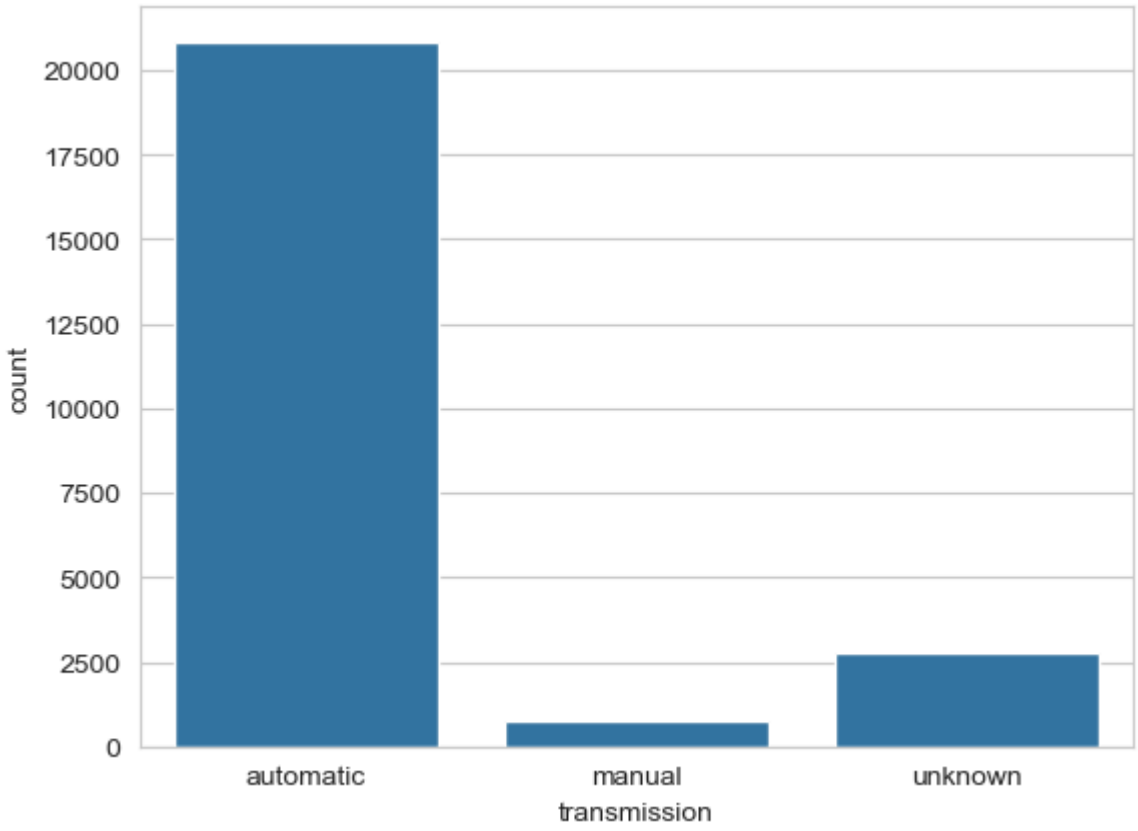
Теперь разберёмся с признаком transmission

In [25]:
```
sns.countplot(data=data_filled_cat, x="transmission")
plt.show()
```



У столбца transmission всего два уникальных значений, причём одно из них встречается в датасете намного чаще другого. Чтобы не увеличивать данный разрыв предлагаю добавить ещё одно значение (категорию) — unknown.

In [26]:
```
# Заполнение признака transmission
data_filled_cat.loc[:,"transmission"], _, _ = impute_column(data_filled_cat, "trans

sns.countplot(data=data_filled_cat, x="transmission")
plt.show()
```



In [27]:
```
# Проверка отсутствия нулевых значений
data_filled_cat.isnull().sum()
```

Out[27]:
```
year            0
make            0
model           0
trim            0
body            0
transmission    0
vin             0
state           0
condition       0
odometer        0
color           0
interior        0
seller          0
mmr             0
sellingprice    0
saledate        0
dtype: int64
```

Кодирование категориальных признаков

In [28]:
```
# Количество уникальных значений в колонках
for column in cat_missing_cols + ["vin", "state", "seller", "saledate"]:
    print (f"Количество уникальных значений в колонке {column}: {len((data_filled_c
```

```
Количество уникальных значений в колонке make: 48
Количество уникальных значений в колонке model: 604
Количество уникальных значений в колонке trim: 911
Количество уникальных значений в колонке body: 62
Количество уникальных значений в колонке transmission: 3
Количество уникальных значений в колонке color: 19
Количество уникальных значений в колонке interior: 17
Количество уникальных значений в колонке vin: 24345
Количество уникальных значений в колонке state: 37
Количество уникальных значений в колонке seller: 3746
Количество уникальных значений в колонке saledate: 1917
```

Обработаем категориальные признаки следующим образом:

1. Удалим столбец с vin, так как для каждой строки он уникален, то есть не несёт полезной информации
2. Преобразуем saledate в datetime
3. Закодируем transmission с помощью OHE
4. Все остальные категориальные признаки закодируем с помощью Frequency Encoder-a

In [29]:
```
# Первые два пункта
data_coded = data_filled_cat.copy()
data_coded = data_coded.drop(columns=["vin"])
data_coded["saledate"] = pd.to_datetime(data_coded["saledate"], format='mixed', utc

data_coded.head()
```

Out[29]:

	year	make	model	trim	body	transmission	state	condition	odometer	col
0	2008	Mercury	Milan	Premier	Sedan	automatic	ny	28.0	84302.0	bla
1	2007	Ford	Fusion	SE	Sedan	automatic	ca	2.0	151270.0	bla
2	2014	Kia	Rio	LX	sedan	automatic	nv	43.0	36563.0	wh
3	2013	Ford	Escape	SEL	suv	automatic	il	44.0	19503.0	bro
4	2013	Mercedes-Benz	C-Class	C250 Sport	Sedan	automatic	ca	29.0	17151.0	wh

In [30]:
```
# Применение OHE для transmission
ce_OHE = OneHotEncoder(cols=["transmission"])
data_coded = ce_OHE.fit_transform(data_coded)
data_coded.head()
```

Out[30]:

	year	make	model	trim	body	transmission_1	transmission_2	transmission_3
0	2008	Mercury	Milan	Premier	Sedan	1	0	0
1	2007	Ford	Fusion	SE	Sedan	1	0	0
2	2014	Kia	Rio	LX	sedan	1	0	0
3	2013	Ford	Escape	SEL	suv	1	0	0
4	2013	Mercedes-Benz	C-Class	C250 Sport	Sedan	1	0	0

In [31]:
```
# Применение Frequency Encoder для остальных категориальных признаков
ce_Freq = CountEncoder(cols=data_coded.select_dtypes(include="object").columns, nor
data_coded = ce_Freq.fit_transform(data_coded)
data_coded.head()
```

Out[31]:

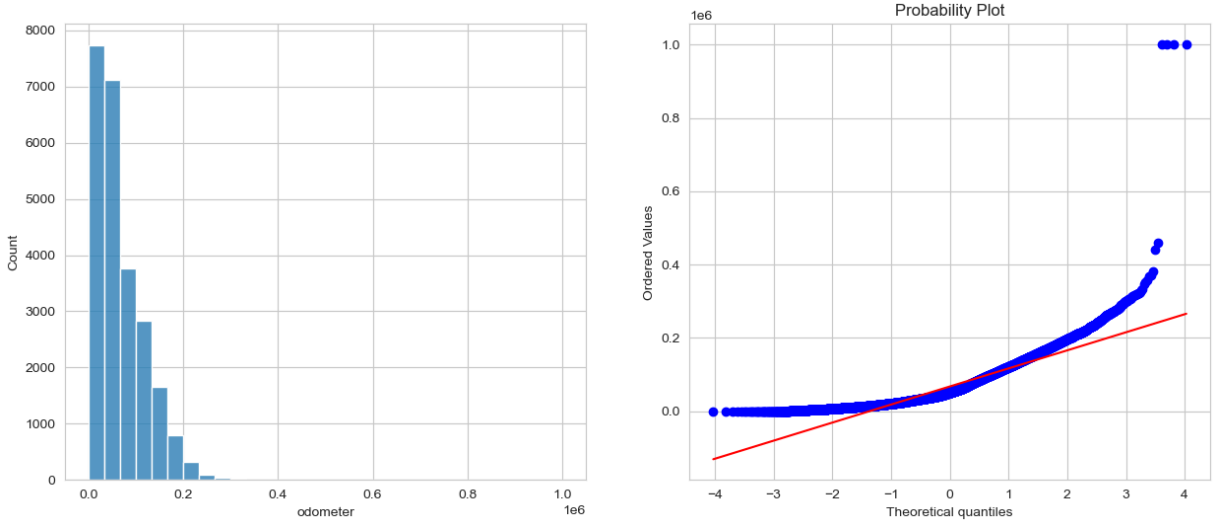
	year	make	model	trim	body	transmission_1	transmission_2	transmissi
0	2008	0.003324	0.000328	0.001067	0.365931	1	0	
1	2007	0.172234	0.023966	0.081254	0.365931	1	0	
2	2014	0.033158	0.002544	0.038862	0.079777	1	0	
3	2013	0.172234	0.022735	0.017236	0.042392	1	0	
4	2013	0.029670	0.008495	0.002298	0.365931	1	0	

Нормализация числовых признаков

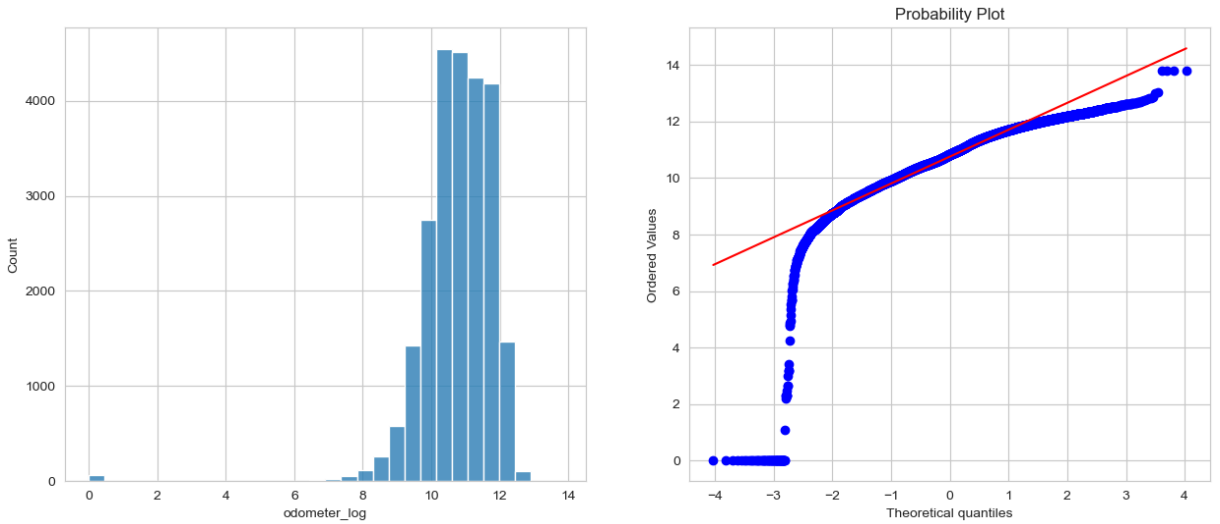
В данном разделе мы рассмотрим три варианта нормализация для поля odometer: логарифм, преобразование Бокса-Кокса, преобразование Йео-Джонсона

In [32]:
```
def diagnostic_plots(df, variable):
    plt.figure(figsize=(15,6))
    # гистограмма
    plt.subplot(1, 2, 1)
    sns.histplot(data=df, x=variable, bins=30)
    ## Q-Q plot
    plt.subplot(1, 2, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.show()
```

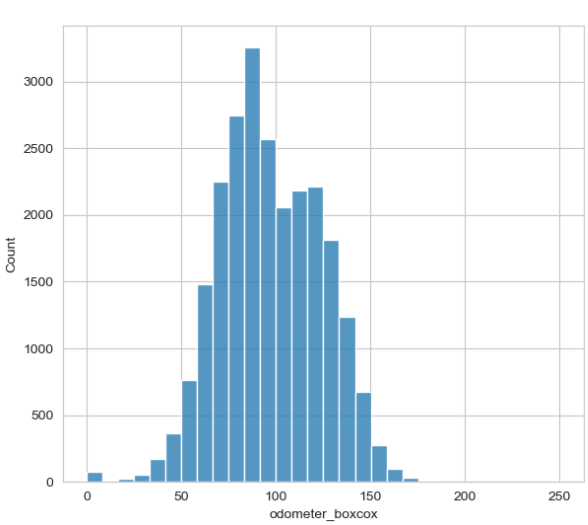
In [33]:
```
# Исходные данные
diagnostic_plots(data_coded, "odometer")
```

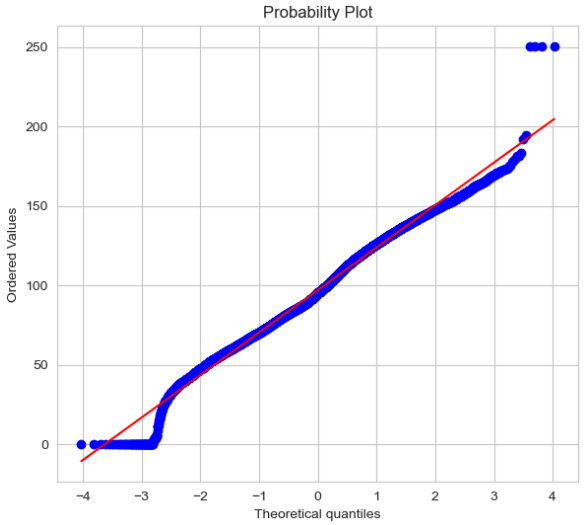


In [34]:
```
# Логарифмическое преобразование
data_coded_norm = data_coded.copy()
data_coded_norm["odometer_log"] = np.log(data_coded_norm["odometer"])
diagnostic_plots(data_coded_norm, "odometer_log")
```



In [35]:
```
# Преобразование Бокса-Кокса
data_coded_norm["odometer_boxcox"], param_boxcox = stats.boxcox(data_coded_norm["od
diagnostic_plots(data_coded_norm, "odometer_boxcox")
```

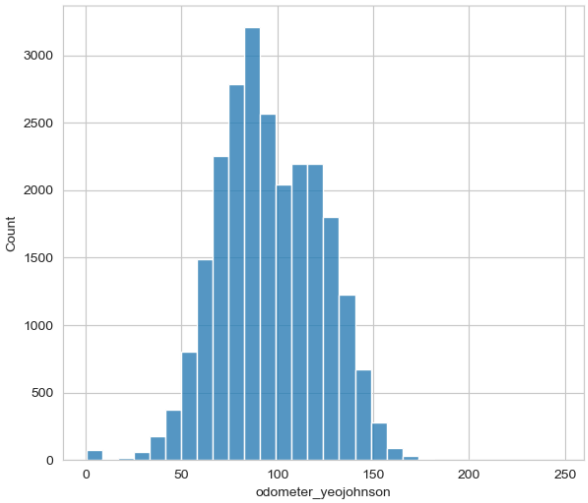


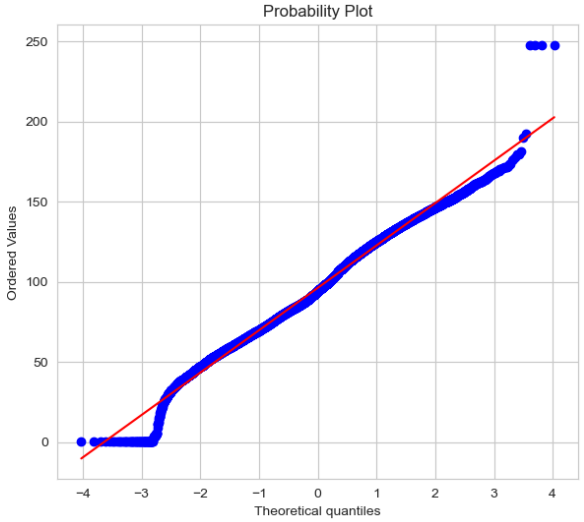


In [36]:
```
param_boxcox
```

Out[36]: 0.3175989307690552

In [37]:
```
# Преобразование Йео-Джонсона
data_coded_norm["odometer_yeojohnson"], param_yeojohnson = stats.yeojohnson(data_co
diagnostic_plots(data_coded_norm, "odometer_yeojohnson")
```





In [38]:
```
param_yeojohnson
```

Out[38]: 0.31654045587180696