

▼ Лабораторная работа №2

"Обработка признаков (часть 1)"

Выполнила: Пинская Н.М.

Группа: ИУ5-21М

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:
 - устранение пропусков в данных;
 - кодирование категориальных признаков;
 - нормализацию числовых признаков.

```
#импортируем библиотеки
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
# from category_encoders.target_encoder import TargetEncoder as ce_TargetEncoder
from sklearn.preprocessing import LabelEncoder
%matplotlib inline
sns.set(style="ticks")
```

```
# Подключение к gogle диску
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive")
```

```
# Вывод содержимого папки на диске
import os
data_root = '/content/drive/MyDrive/MMO'
print(os.listdir(data_root))
```

```
['pulitzer-circulation-data.csv', 'mmsa-icu-beds2.csv', 'movies.csv.zip', 'movies2.csv.zip', 'movies1.csv.zip']
```

► Устранение пропусков в данных

[] ↳ 21 cells hidden

Кодирование категориальных признаков

```
data_code = pd.read_csv('/content/drive/MyDrive/MMO/laptop.csv', sep=",")
```

```
data_code.head()
```

	Unnamed: 0	Brand	Model	Series	Processor	Processor_Gen	RAM	Hard_Disk_Capacity	OS	Rating
0	0	DELL	Inspiron	NaN	i3	11th	8.0	1 TB HDD	Windows 11 Home	3.7
1	1	DELL	Vostro	NaN	i5	11th	8.0	1 TB HDD	Windows 10 Home	3.6
2	2	ASUS	VivoBook	15	i3	10th	8.0	512 GB SSD	Windows 10 Home	4.3

```
!pip install category_encoders
from category_encoders.target_encoder import TargetEncoder as ce_TargetEncoder
data_coded = ce_TargetEncoder().fit_transform(data_code[data_code.columns.difference(['Brand', 'Model', 'RAM', 'OS', 'Rating'])])
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: category_encoders in /usr/local/lib/python3.7/dist-packages (2.4.1)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from category_encoders)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1->category_encoders)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.1->category_encoders)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders)
```

```
data_coded
```

	Hard_Disk_Capacity	OS	Price	Processor	Processor_Gen	Rating	Series	Unnamed: 0
0	8.093023	8.181818	39040	7.379310	9.507246	3.7	8.320000	0
1	8.093023	8.869565	50840	10.777778	9.507246	3.6	8.320000	1
2	10.000000	8.181818	37940	7.379310	7.666667	4.3	7.001479	2
3	8.093023	9.999997	44440	7.379310	9.507246	4.4	8.320000	3
4	10.000000	8.869565	57940	10.777778	7.666667	4.5	11.421634	4
...
125	8.093023	8.181818	39040	7.379310	9.507246	3.7	8.320000	125
126	8.093023	4.868760	32905	7.379310	5.243303	4.2	8.320000	126
127	8.093023	4.868760	39949	7.379310	7.666667	3.1	8.320000	127
128	8.093023	8.869565	42750	7.379310	9.507246	2.9	7.001479	128
129	7.200000	8.181818	39940	7.379310	9.507246	4.3	8.000000	129

130 rows × 8 columns

```

data_code['OS'].unique()

array(['Windows 11 Home', 'Windows 10 Home', '256 GB SSD', 'Windows 10',
      'DOS', nan, 'Chrome OS', '128 GB SSD'], dtype=object)

data_code['Processor'].unique()

array(['i3', 'i5', '3250U', '3050U', '5600U', '-', '5500U', 'i7', '5300U',
      '5600H', '3450U', nan, '4800HS', '4600H', '4800H', 'R3-3250U',
      'AMD', 'A9'], dtype=object)

data_coded['Processor'].unique()

array([ 7.37931034, 10.77777778,  8.07425756,  8.62295082,  5.65437143,
        8.16753728, 13.74498875,          nan])

data_coded['OS'].unique()

array([ 8.18181818,  8.86956522,  9.99999689, 10.39978072,  4.8687604 ,
        nan,  5.24330296,  8.62295082])

#закодировать лейблами категории и конкатенировать
le = LabelEncoder()
category_le = le.fit_transform(data_code['Hard_Disk_Capacity'])

np.unique(category_le)


array([0, 1, 2, 3, 4, 5, 6, 7])

#преобразуем полученный массив в колонку
data_tempor = np.array(category_le)
data_le = pd.DataFrame(data_tempor, columns=['category_le'])

#объединяем колонки оригинальной таблицы с кодированными колонками
data_result = pd.concat([data_le['category_le'], data_code.iloc[:, 1:4], data_coded['Processor'], data_code[

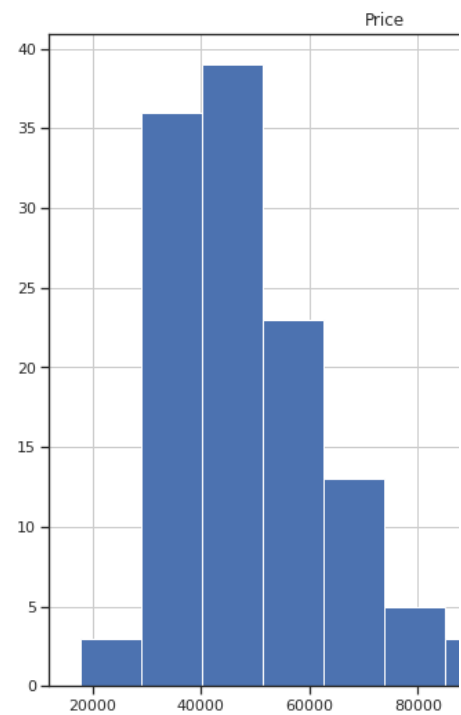
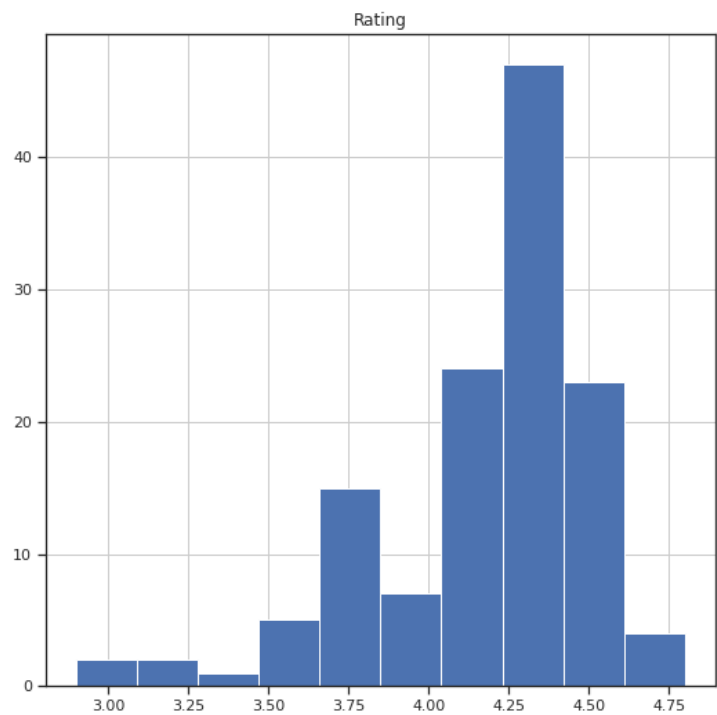
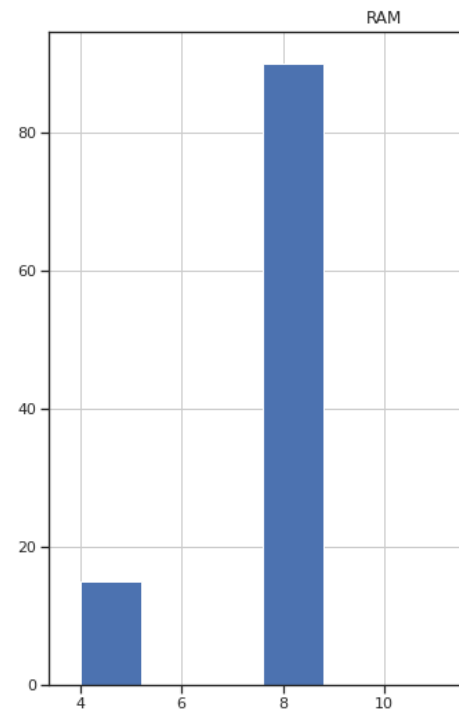
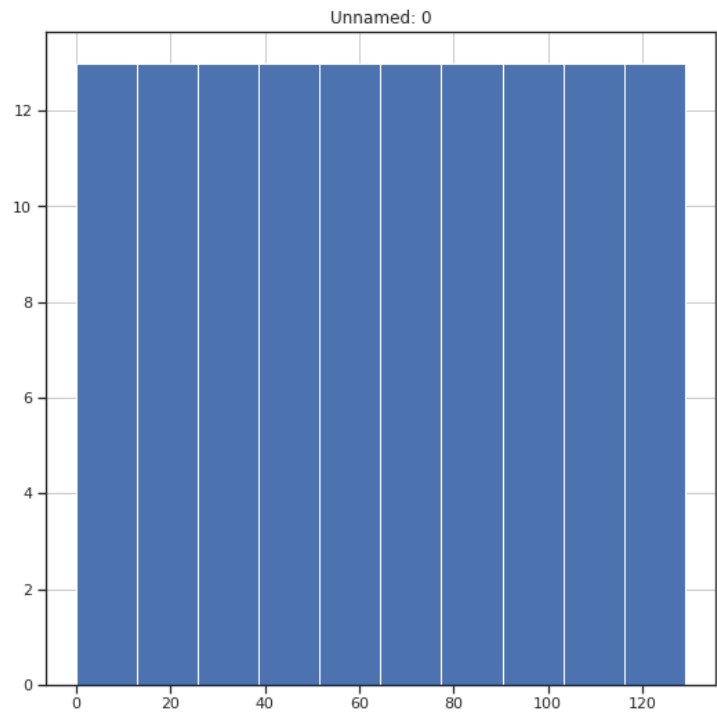
#итоговая таблица с кодированными категориальными признаками.
data_result.head(10)

```

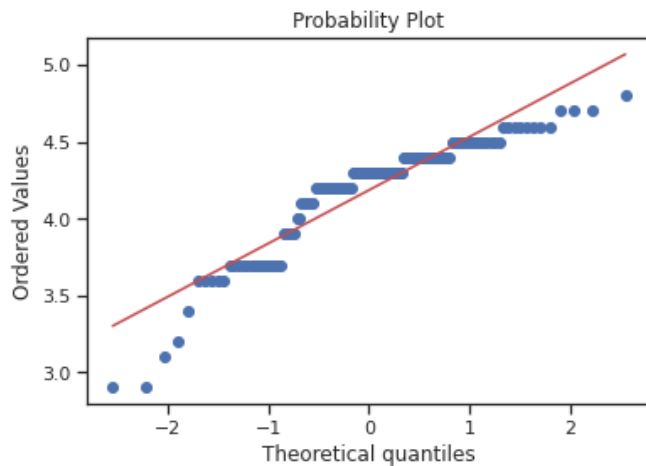
	category_le	Brand	Model	Series	Processor	Price	OS	Model	RAM	
0	0	DELL	Inspiron	NaN	7.379310	39040	8.181818	Inspiron	8.0	
1	0	DELL	Vostro	NaN	10.777778	50840	8.869565	Vostro	8.0	
2	5	ASUS	VivoBook	15	7.379310	37940	8.181818	VivoBook	8.0	
3	0	DELL	Inspiron	NaN	7.379310	44440	9.999997	Inspiron	8.0	
4	5	ASUS	TUF	Gaming	10.777778	57940	8.869565	TUF	8.0	
5	3	ASUS	Ryzen	3	8.074258	35940	8.869565	Ryzen	8.0	
6	3	DELL	Inspiron	Athlon	8.622951	33940	8.181818	Inspiron	4.0	
7	5	DELL	Inspiron	NaN	10.777778	69040	10.399781	Inspiron	16.0	
8	0	Lenovo	IdeaPad	3	7.379310	37440	8.869565	IdeaPad	8.0	
9	5	HP	NaN	NaN	10.777778	56449	8.181818	NaN	8.0	

▼ Нормализация числовых признаков

```
#нормализуем признак из предыдущего датасета
skip.hist(figsize=(20,20))
plt.show()
```



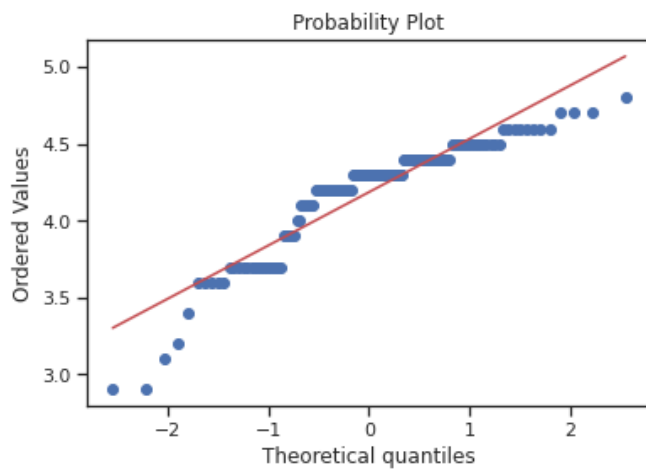
```
stats.probplot(skip['Rating'], dist="norm", plot = plt)
plt.show()
```



```
#применяем преобразование Бокса-Кокса.
skip['Rating_bxcox'], param = stats.boxcox(skip['Rating'])
print('Оптимальное значение  $\lambda$  = {}'.format(param))
```

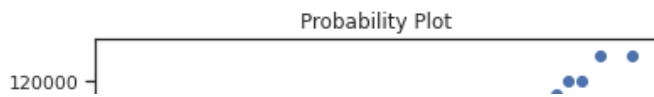
```
stats.probplot(skip['Rating'], dist="norm", plot = plt)
plt.show()
```

Оптимальное значение λ = 6.19020443917887



Результат, в целом, плохой, но учитывая степень отклонения исходного распределения от нормальности, ситуация улучшилась.

```
stats.probplot(skip['Price'], dist="norm", plot = plt)
plt.show()
```



```
# применяем логарифмическое преобразование
#получаем достаточно неплохой результат
skip['Price_log'] = np.log(skip['Price'])

stats.probplot(skip['Price_log'], dist="norm", plot = plt)
plt.show()
```

