

Лабораторная работа №2 по  
дисциплине  
«Методы машинного обучения» на  
тему  
«Обработка пропусков в данных, кодирование  
категориальных признаков, Нормализация числовых  
признаков, масштабирование данных»

Выполнил:  
Студент группы ИУ5-21М Ся  
Бэйбэй

---

---

## 1. Цель лабораторной работы

Изучить способы предварительной обработки данных для дальнейшего формирования моделей [1].

## 2. Задание

Требуется [1]:

1. Выбрать набор данных (датасет), содержащий категориальные признаки и пропуски в данных. Для выполнения следующих пунктов можно использовать несколько различных наборов данных.
2. Для выбранного датасета (датасетов) на основе материалов лекции решить следующие задачи:
  - обработку пропусков в данных;
  - кодирование категориальных признаков;
  - масштабирование данных.

## 3. Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков :

```
In [50]: import numpy as np
import pandas as pd
import seaborn as sns
import sklearn.impute
import sklearn.preprocessing
%matplotlib inline
sns.set(style="ticks")
```

Зададим ширину текстового представления данных, чтобы в дальнейшем текст в отчёте влезал на А4 :

```
In [2]: pd.set_option("display.width", 70)
```

Для выполнения данной лабораторной работы возьмём набор данных по приложениям в Covid\_19 Data:

```
In [3]: data = pd.read_csv("covid_19_clean_complete_2022.csv")
```

Посмотрим на эти наборы данных:

```
In [5]: data.dtypes
```

```
In [9]: data.dtypes
```

```
Out[9]: Province/State    object
Country/Region          object
Lat                     float64
Long                    float64
Date                    object
Confirmed                int64
Deaths                  int64
Recovered                int64
Active                  int64
WHO Region              object
dtype: object
```

```
In [10]: data.shape
```

```
Out[10]: (214894, 10)
```

```
In [11]: data.isnull().sum()
```

```
Out[11]: Province/State    149189
Country/Region            0
Lat                       1546
Long                      1546
Date                      0
Confirmed                  0
Deaths                    0
Recovered                  0
Active                     0
WHO Region                 2319
dtype: int64
```

### 3.1. Обработка пропусков в данных

Найдем все пропуски в данных:

```
In [11]: data.isnull().sum()
```

```
Out[11]: Province/State    149189
Country/Region            0
Lat                       1546
Long                      1546
Date                      0
Confirmed                  0
Deaths                    0
Recovered                  0
Active                     0
WHO Region                 2319
dtype: int64
```

```
In [15]: hcols_with_na = [c for c in data.columns if data[c].isnull().sum() > 0]
[(c, data[c].isnull().sum()) for c in hcols_with_na]
[(c, data[c].isnull().mean()) for c in hcols_with_na]
```

```
Out[15]: [('Province/State', 0.6942446043165468),
('Lat', 0.007194244604316547),
('Long', 0.007194244604316547),
('WHO Region', 0.01079136690647482)]
```

Очевидно, что мы выпустим с колонкой Province/State.

```
In [19]: data_de = data.drop(["Province/State"], axis=1)
```

```
In [20]: data_de.dtypes
```

```
Out[20]: Country/Region    object
Lat                       float64
Long                      float64
Date                      object
Confirmed                 int64
Deaths                   int64
Recovered                 int64
Active                   int64
WHO Region                object
dtype: object
```

Очевидно, что мы будем работать с колонкой `Lat`.

```
In [58]: from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```

```
In [59]: 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 [63]: 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,))

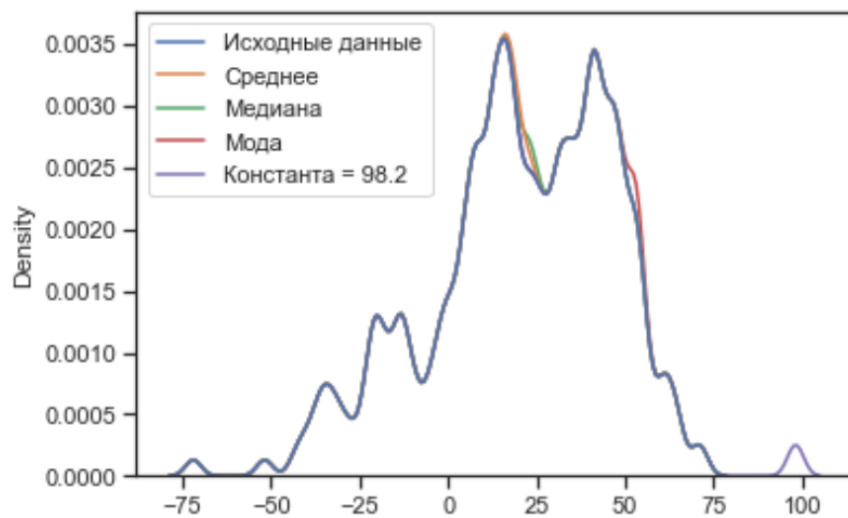
    new_df = pd.DataFrame({'Исходные данные': original_data})

    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 [69]: research_impute_numeric_column(data_de, "Lat", 98.2)
```

```
In [69]: research_impute_numeric_column(data_de, "Lat", 98.2)
```



Очевидно, что использование этих методов приводит к незначительным различиям в конечных результатах. Но я думаю, что метод “Среднее” относительно хороший.

### 3.2. Кодирование категориальных признаков

Рассмотрим колонку Type:

```
In [117]: types = data_de["WHO Region"].dropna().astype(str)
types.value_counts()
```

```
Out[117]: Europe          64159
Western Pacific    50245
Americas          37104
Africa            36331
Eastern Mediterranean 17006
South-East Asia    7730
Name: WHO Region, dtype: int64
```

Выполним кодирование категорий целочисленными значениями:

```
In [128]: from category_encoders.count import CountEncoder as ce_CountEncoder
          from category_encoders.target_encoder import TargetEncoder as ce_TargetEncoder

In [126]: ce_CountEncoder2 = ce_CountEncoder(normalize=True)
          data_FREQ_ENC = ce_CountEncoder2.fit_transform(data_de[data_de.columns.difference(['Survived'])])
          data_FREQ_ENC

Out[126]:
```

|        | Active | Confirmed | Country/Region | Date     | Deaths | Lat        | Long       | Recovered | WHO Region |
|--------|--------|-----------|----------------|----------|--------|------------|------------|-----------|------------|
| 0      | 0      | 0         | 0.003597       | 0.001294 | 0      | 33.939110  | 67.709953  | 0         | 0.079137   |
| 1      | 0      | 0         | 0.003597       | 0.001294 | 0      | 41.153300  | 20.168300  | 0         | 0.298561   |
| 2      | 0      | 0         | 0.003597       | 0.001294 | 0      | 28.033900  | 1.659600   | 0         | 0.169065   |
| 3      | 0      | 0         | 0.003597       | 0.001294 | 0      | 42.506300  | 1.521800   | 0         | 0.298561   |
| 4      | 0      | 0         | 0.003597       | 0.001294 | 0      | -11.202700 | 17.873900  | 0         | 0.169065   |
| ...    | ...    | ...       | ...            | ...      | ...    | ...        | ...        | ...       | ...        |
| 214889 | 644414 | 649971    | 0.003597       | 0.001294 | 5557   | 31.952200  | 35.233200  | 0         | 0.079137   |
| 214890 | 530    | 530       | 0.003597       | 0.001294 | 0      | 39.904200  | 116.407400 | 0         | 0.010791   |
| 214891 | 9639   | 11774     | 0.003597       | 0.001294 | 2135   | 15.552727  | 48.516388  | 0         | 0.079137   |
| 214892 | 309655 | 313613    | 0.003597       | 0.001294 | 3958   | -13.133897 | 27.849332  | 0         | 0.169065   |
| 214893 | 233342 | 238739    | 0.003597       | 0.001294 | 5397   | -19.015438 | 29.154857  | 0         | 0.169065   |

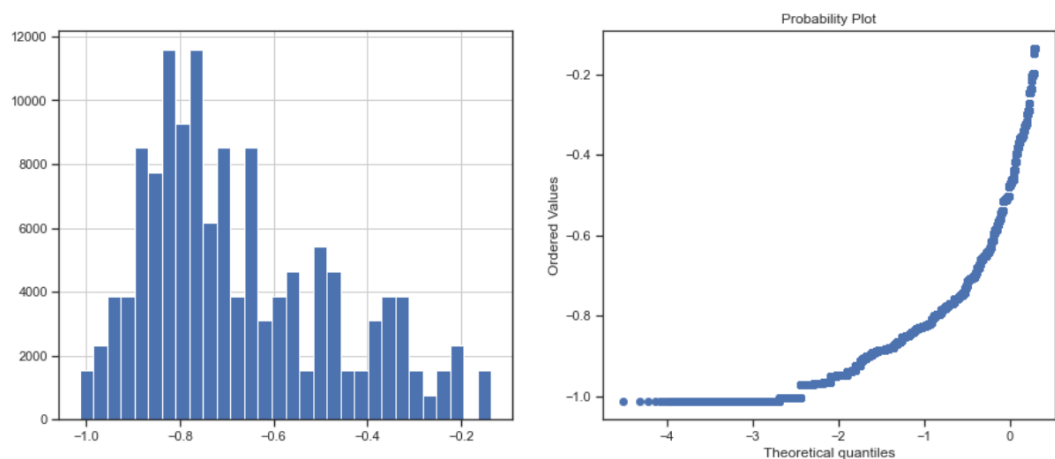
214894 rows x 9 columns

```
In [125]: data_FREQ_ENC['WHO Region'].unique()

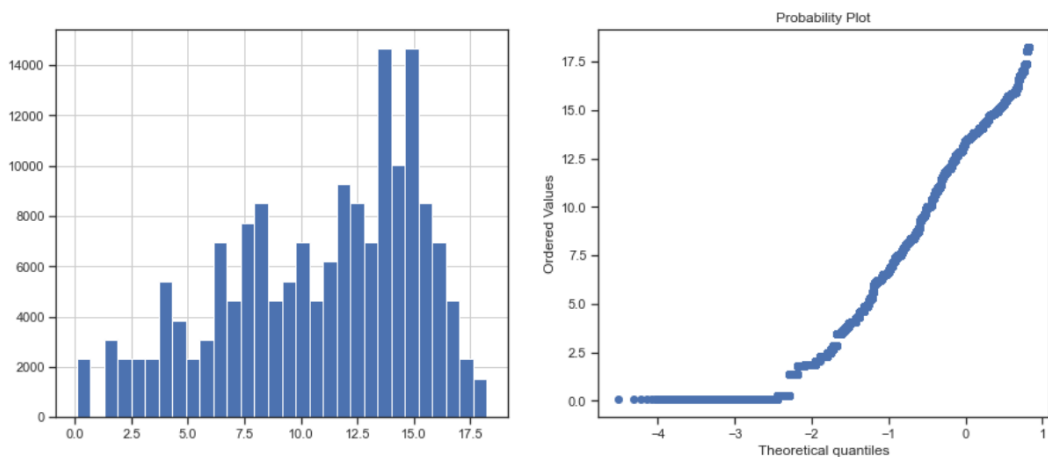
Out[125]: array([0.07913669, 0.29856115, 0.16906475, 0.01079137, 0.17266187,
                0.23381295, 0.03597122])
```

### 3.3. Нормализация числовых признаков

```
In [34]: data['Lat'] = 1 / (data['Lat'])
          diagnostic_plots(data, 'Lat')
```



```
In [39]: data_de['Lat'] = data_de['Lat']**(2)
          diagnostic_plots(data_de, 'Lat')
```

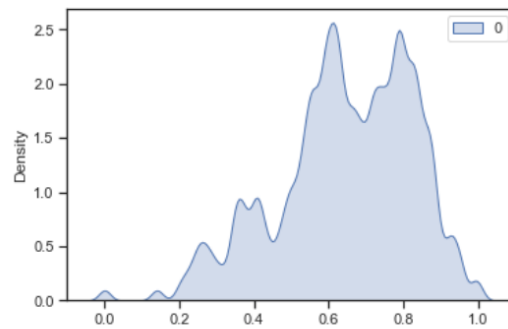


### 3.4. Масштабирование данных

Для начала попробуем обычное MinMax-масштабирование и StandardScaler:

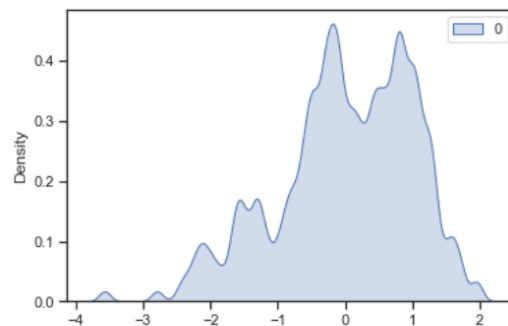
```
In [153]: mm = sklearn.preprocessing.MinMaxScaler()
sns.kdeplot(data=mm.fit_transform(data_de[["Lat"]]), shade=True, color="g")
```

Out[153]: <AxesSubplot:ylabel='Density'>



```
In [154]: mm = sklearn.preprocessing.StandardScaler()
sns.kdeplot(data=mm.fit_transform(data_de[["Lat"]]), shade=True, color="g")
```

Out[154]: <AxesSubplot:ylabel='Density'>

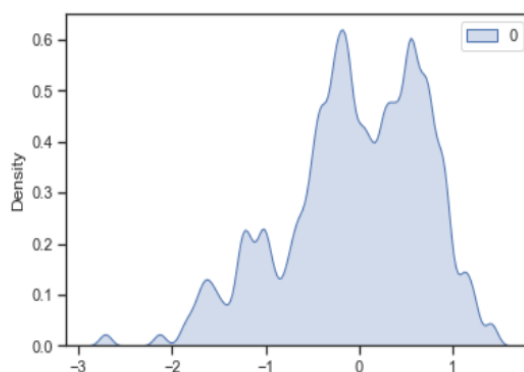


```
In [155]: mm = sklearn.preprocessing.RobustScaler()
sns.kdeplot(data=mm.fit_transform(data_de[["Lat"]]), shade=True, color="g")
```

Результат вполне ожидаемый и вполне приемлемый. Но попробуем и другие варианты, например, масштабирование на основе RobustScaler:

```
In [155]: mm = sklearn.preprocessing.RobustScaler()
sns.kdeplot(data=mm.fit_transform(data_de[["Lat"]]), shade=True, color="g")
```

Out[155]: <AxesSubplot:ylabel='Density'>



Также результат ожидаемый, но его применимость зависит от дальнейшего использования.

## **Список литературы**

[1] COVID -19 Global Reports early March 2022 [Electronic resource] // Kaggle. — 2022. — Access mode: <https://www.kaggle.com/danielfesalbon/covid-19-global-reports-early-march-2022> (online; accessed:16.03.2022).