Лабораторная работа №5

Разведочный анализ и предобработка данных

Цель: изучить особенности предварительной обработки исходных данных, преобразования качественных показателей в количественный вид.

1. Импорт набора данных Mushroom Data Set

Исследуем набор данных Mushroom Data Set, предназначенный для проведения бинарной классификации грибов. Набор данных включает 8124 наблюдений по 22 показателям. То есть каждое наблюдение представляет собой отдельный гриб, характеризующийся 22 показателями. Каждый показатель является качественным, то есть включает только буквенные обозначения. Каждый гриб классифицируется на 2 класса: ядовитый гриб (метка p) и неядовитый гриб (метка e).

Набор данных можно скачать с сайта школы Donald Bren School of Information and Computer Sciences: https://archive.ics.uci.edu/ml/datasets/Mushroom

На веб-странице нажмем «Data Folder»:

Data Set Characteristics:	Multivariate	**Number of Instances:**	8124	**Area:**	Life
Attribute Characteristics:	Categorical	**Number of Attributes:**	22	**Date Donated**	1987-04-27
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	801248

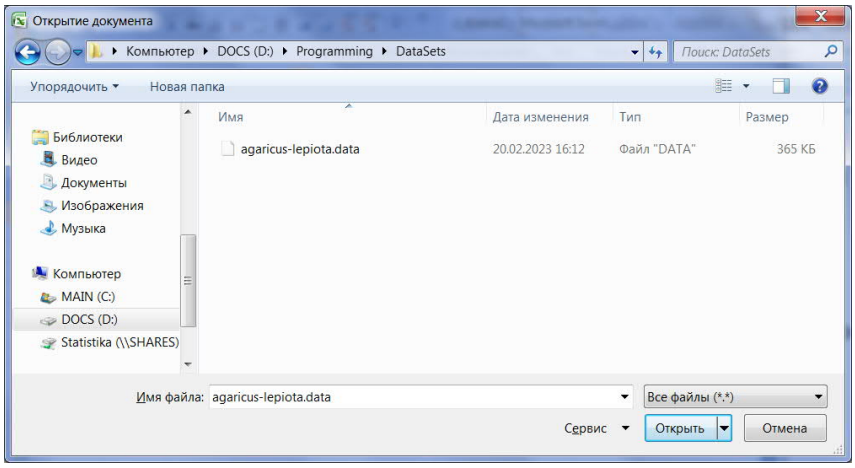
Далее нажмем на «agaricus-lepiota.data», чтобы скачать одноименный файл с данными:

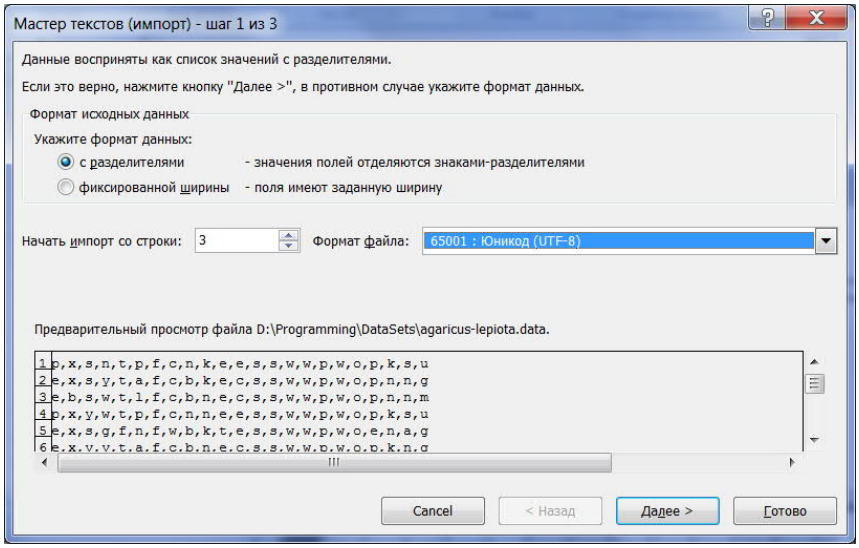
Index of /ml/machine-learning-databases/mushroom

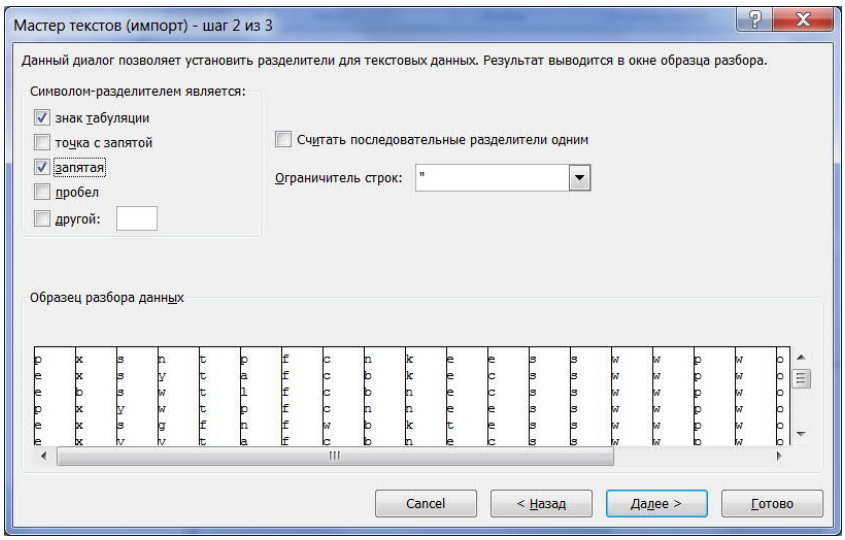
- Parent Directory
- Index
- README
- agaricus-lepiota.data
- agaricus-lepiota.names
- expanded.Z

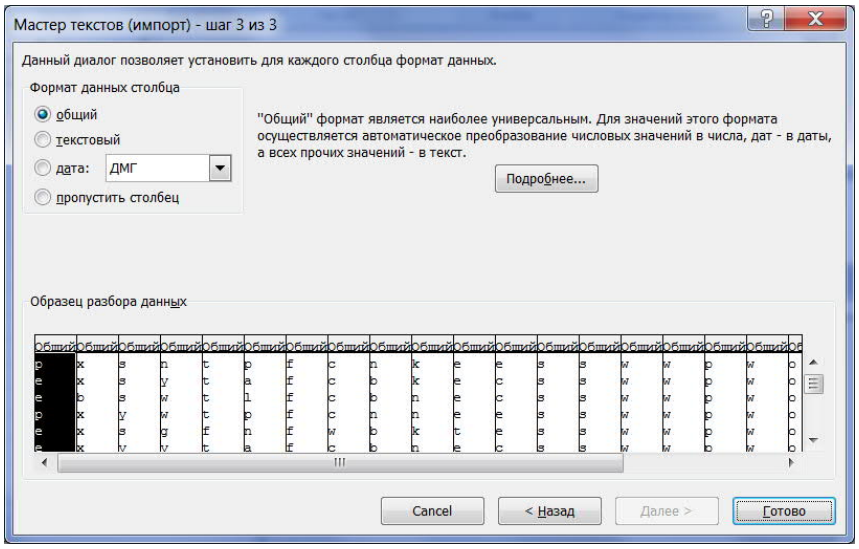
Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips SVN/1.7.14 Phusion_Passenger/4.0.53 mod_perl/2.0.11 Perl/v5.16.3 Server at archive.ics.uci.edu Port 443

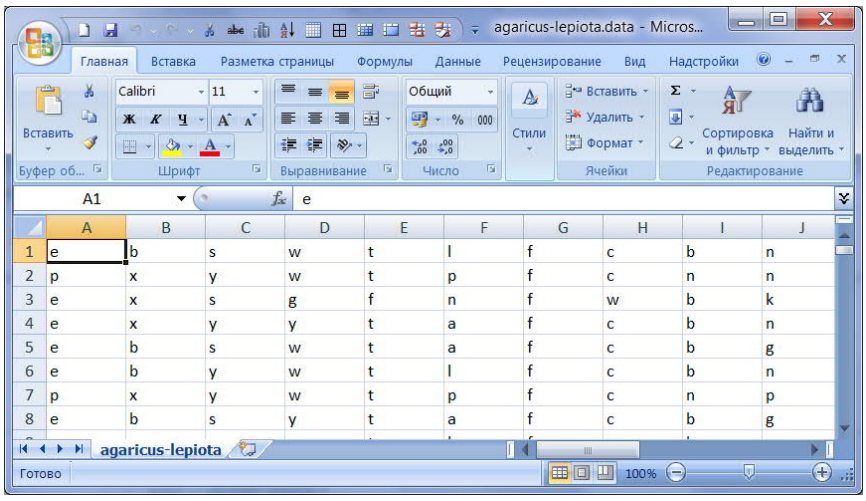
Скачанный файл имеет расширение .data. Преобразуем его в файл с расширением .csv. Для этого откроем файл в Excel и с помощью инструмента «Мастер текстов» импортируем данные, преобразовав их в табличный вид:



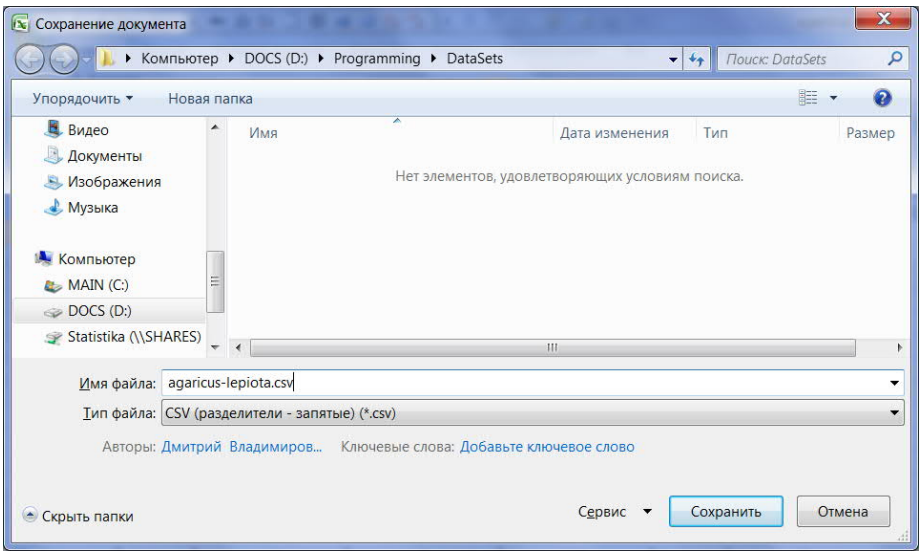








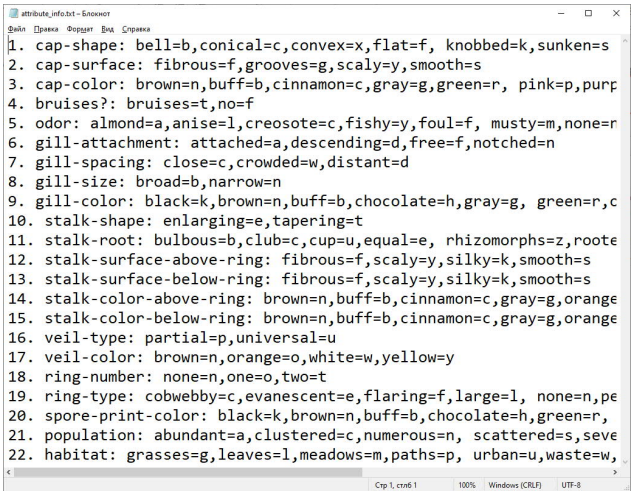
Сохраним файл под именем agaricus-lepiota.csv:



Далее необходимо добавить названия столбцов. На сайте представлено следующее описание 22 показателей:

```
Attribute Information:
1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,
pink=p,purple=u,red=e,white=w,yellow=y
4. bruises?: bruises=t,no=f
5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6. gill-attachment: attached=a,descending=d,free=f,notched=n
7. gill-spacing: close=c,crowded=w,distant=d
8. gill-size: broad=b,narrow=n
9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g,
green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
10. stalk-shape: enlarging=e,tapering=t
11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,
pink=p,red=e,white=w,yellow=y
15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,
pink=p,red=e,white=w,yellow=y
16. veil-type: partial=p,universal=u
17. veil-color: brown=n,orange=o,white=w,yellow=y
18. ring-number: none=n,one=o,two=t
19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,
none=n,pendant=p,sheathing=s,zone=z
20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,
orange=o,purple=u,white=w,yellow=y
21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d
```

Создадим скрипт для извлечения имен показателей из данного текста (листинг 1). Для этого сохраним текст в файл attribute_info.txt.



При этом в данном тексте не указано название результативного показателя – сам класс, к которому относится тот или иной гриб. Установим следующее название для данного показателя: «class».

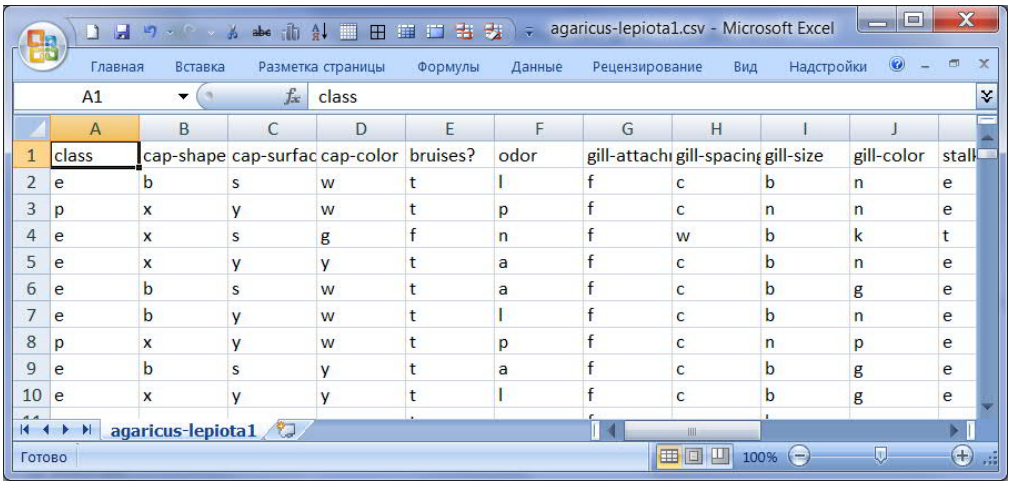
Листинг 1. Программа для извлечения названий столбцов из информации о показателях набора данных Mushroom Data Set

```
# Подключение библиотек
import os               # Работа с ОС
import pandas as pd     # Таблицы DataFrame

# Пути к файлам
cwd = os.getcwd()                                       # Текущая директория
filePath = cwd + '\\' + 'attribute_info.txt'            # Файл с текстом
filePath_csv = cwd + '\\' + 'agaricus-lepiota.csv'    # Набор данных без имен столбцов
filePath_csv1 = cwd + '\\' + 'agaricus-lepiota1.csv' # Набор данных с именами столбцов


#
# 1. Извлечение названий показателей
#

# Открытие файла с информацией о показателях
with open(filePath, 'r') as f:
    file_text = []
    for line in f:
        file_text.append(line)

# Извлечение названий показателей
attributes_names = ['class']
for atr in file_text:
    name = atr[: atr.find(':')]
    name = name[atr.find('. ') + 2 :]
    attributes_names.append(name)


#
# 2. Добавление названий столбцов в файл CSV
#

# Открытие файла csv
data_csv = pd.read_csv(filePath_csv,
                       sep = ';',
                       header = None)
# Установка названий столбцов
data_csv.columns = attributes_names
# Сохранение DataFrame в файл CSV
data_csv.to_csv(filePath_csv1,
                sep = ';',
                index = False)
```

В результате работы программы формируется список названий показателей `attributes_names`:

```
In [31]: attributes_names
Out[31]:
['class',
 'cap-shape',
 'cap-surface',
 'cap-color',
 'bruises?',
 'odor',
 'gill-attachment',
 'gill-spacing',
 'gill-size',
 'gill-color',
 'stalk-shape',
 'stalk-root',
```

Далее с помощью библиотеки pandas импортируется набор исходных данных из файла agaricus-lepiota.csv и представляется как объект DataFrame. Список attributes_names устанавливается в качестве названий столбцов и объект DataFrame сохраняется в отдельный файл agaricus-lepiota1.csv.



	A	B	C	D	E	F	G	H	I	J
1	class	cap-shape	cap-surfac	cap-color	bruises?	odor	gill-attachi	gill-spacing	gill-size	gill-color	stalk
2	e	b	s	w	t	l	f	c	b	n	e
3	p	x	y	w	t	p	f	c	n	n	e
4	e	x	s	g	f	n	f	w	b	k	t
5	e	x	y	y	t	a	f	c	b	n	e
6	e	b	s	w	t	a	f	c	b	g	e
7	e	b	y	w	t	l	f	c	b	n	e
8	p	x	y	w	t	p	f	c	n	p	e
9	e	b	s	y	t	a	f	c	b	g	e
10	e	x	y	y	t	l	f	c	b	g	e

Создадим новый скрипт и импортируем набор данных с установленными названиями столбцов:

```
# Подключение библиотек
import os                # Работа с ОС
import pandas as pd      # Таблицы DataFrame

# Пути к файлам
cwd = os.getcwd()                                    # Текущая директория
filePath = cwd + '\\' + 'agaricus-lepiota1.csv' # Набор данных с именами столбцов

# Импорт данных
df = pd.read_csv(filePath,
                 sep = ';',
                 header = 0)
```

2. Предобработка данных

2.1 Уникальные значения столбцов

Определим число уникальных значений каждого показателя, чтобы выяснить:

- какие столбцы имеют по всем строкам одинаковое значение и удалить их, так как они не несут в себе никакой полезной информации с точки зрения анализа данных;
- какие столбцы имеют символы пропущенных значений, например: «?», NaN, NULL, None и т.д.

Получить массив уникальных значений столбца можно с помощью метода pandas.Series.unique(). Объектом Series является каждый столбец DataFrame, если обратится к нему через индексы следующим образом:

```
таблица.iloc[:, индекс_столбца]
```

```
In [32]: df
Out[32]:
     class cap-shape cap-surface  ... spore-print-color population habitat
0        e         b           s  ...                 n          n       m
1        p         x           y  ...                 k          s       u
2        e         x           s  ...                 n          a       g
3        e         x           y  ...                 k          n       g
4        e         b           s  ...                 k          n       m
...    ...       ...         ...  ...               ...        ...     ...
8117     e         k           s  ...                 b          c       l
8118     e         x           s  ...                 b          v       l
8119     e         f           s  ...                 b          c       l
8120     p         k           y  ...                 w          v       l
8121     e         x           s  ...                 o          c       l

[8122 rows x 23 columns]

In [33]: df.iloc[:, 0]
Out[33]:
0       e
1       p
2       e
3       e
4       e
       ..
8117    e
8118    e
8119    e
8120    p
8121    e
Name: class, Length: 8122, dtype: object

In [34]: type(df.iloc[:, 0])
Out[34]: pandas.core.series.Series

In [35]: df.iloc[:, 0].unique()
Out[35]: array(['e', 'p'], dtype=object)
```

Как видно, столбец class с индексом 0 имеет два уникальных значения: e, p.

С помощью цикла выведем в консоль уникальные значения всех столбцов:

```
Уникальные значения столбцов:
class ['e' 'p']
cap-shape ['b' 'x' 's' 'f' 'k' 'c']
cap-surface ['s' 'y' 'f' 'g']
cap-color ['w' 'g' 'y' 'n' 'e' 'p' 'b' 'u' 'c' 'r']
bruises? ['t' 'f']
odor ['l' 'p' 'n' 'a' 'f' 'c' 'y' 's' 'm']
gill-attachment ['f' 'a']
gill-spacing ['c' 'w']
gill-size ['b' 'n']
gill-color ['n' 'k' 'g' 'p' 'w' 'h' 'u' 'e' 'b' 'r' 'y' 'o']
stalk-shape ['e' 't']
stalk-root ['c' 'e' 'b' 'r' '?']
stalk-surface-above-ring ['s' 'f' 'k' 'y']
stalk-surface-below-ring ['s' 'f' 'y' 'k']
stalk-color-above-ring ['w' 'g' 'p' 'n' 'b' 'e' 'o' 'c' 'y']
stalk-color-below-ring ['w' 'p' 'g' 'b' 'n' 'e' 'y' 'o' 'c']
veil-type ['p']
veil-color ['w' 'n' 'o' 'y']
ring-number ['o' 't' 'n']
ring-type ['p' 'e' 'l' 'f' 'n']
spore-print-color ['n' 'k' 'u' 'h' 'w' 'r' 'o' 'y' 'b']
population ['n' 's' 'a' 'v' 'y' 'c']
habitat ['m' 'u' 'g' 'd' 'p' 'w' 'l']
```

Выведем количество уникальных значений для каждого столбца:

```
Количество уникальных значений столбцов:
class 2
cap-shape 6
cap-surface 4
cap-color 10
bruises? 2
odor 9
gill-attachment 2
gill-spacing 2
gill-size 2
gill-color 12
stalk-shape 2
stalk-root 5
stalk-surface-above-ring 4
stalk-surface-below-ring 4
stalk-color-above-ring 9
stalk-color-below-ring 9
veil-type 1
veil-color 4
ring-number 3
ring-type 5
spore-print-color 9
population 6
habitat 7

Столбцы с одним уникальным значением:
veil-type
```

Так как столбец veil-type имеет лишь одно уникальное значение, то его следует удалить.

Удалить столбцы с одним уникальным значением можно с помощью метода `drop()`, принимающего в качестве параметра название столбца и индекс оси (`axis`) – 0 для удаления строки и 1 для удаления столбца:

```
# Удаление столбцов с одним уникальным значением
print('\nУдаление столбцов с одним уникальным значением')
for col in df.columns:
    if len(df[col].unique()) == 1:          # Если массив из 1 элемента
        print(f'Удаление столбца: {col}')
        df = df.drop(labels = col, axis = 1) # Удаление
```

```
Удаление столбцов с одним уникальным значением
Удаление столбца: veil-type
```

2.2 Поиск пропущенных значений

Автоматизировать процесс поиска пропущенных значений бывает затруднительно в связи с разными символами пропущенных значений в каждом наборе данных. В рассматриваемом наборе данных таким символом является «?».

Выведем список столбцов с символом пропущенных значений «?»:

```
print('\nНаличие пропущенных значений в столбцах:')
for col in df.columns:
    if '?' in df[col].unique():
        print(col, df[col].unique())
```

```
Наличие пропущенных значений в столбцах:
stalk-root ['c' 'e' 'b' 'r' '?']
```

В нашем случае только столбец «stalk-root» имеет пропущенные значения.

Подсчитаем количество пропущенных значений. Для этого необходимо взять не весь столбец таблицы `DataFrame`, а лишь те ячейки, которые удовлетворяют условию.

Отбор значений столбца по условию осуществляется следующим образом:

```
таблица[столбец][условие]
```

Так как в нашем случае в условии необходимо указать соответствие каждого значения столбца символу «?», то команда приобретает следующий вид:

```
таблица[столбец][таблица[столбец] == '?']
```

Подсчитать число элементов полученного массива можно с помощью функции `len()`:

```
In [55]: df['stalk-root']
Out[55]:
0       c
1       e
2       e
3       c
4       c
       ..
8117    ?
8118    ?
8119    ?
8120    ?
8121    ?
Name: stalk-root, Length: 8122, dtype: object

In [56]: df['stalk-root'][df['stalk-root'] == '?']
Out[56]:
3982    ?
4021    ?
4074    ?
4098    ?
4102    ?
       ..
8117    ?
8118    ?
8119    ?
8120    ?
8121    ?
Name: stalk-root, Length: 2480, dtype: object

In [57]: len(df['stalk-root'][df['stalk-root'] == '?'])
Out[57]: 2480
```

Таким образом мы можем вывести список столбцов с пропущенными значениями и число пропущенных значений:

```
Наличие пропущенных значений в столбцах:
stalk-root ['c' 'e' 'b' 'r' '?'] 2480 (30.53%)
```

2.3 Обработка пропущенных значений

Существуют разные **стратегии обработки пропущенных значений**, зависящие от исследуемого набора данных:

- Если строк с пропущенными значениями **несколько процентов** от общего числа строк в наборе данных, то такие строки удаляют.
- Если строка или столбец имеет **100%** пропущенных значений, то такой столбец или строку удаляют.
- Если пропущенных значений около **50%** от общего числа значений в столбце, то удаление таких строк приведет к потере информации об исходных данных. Тогда такие значения заменяются на средние, либо на значения из соседних ячеек.
- В отдельных случаях пропущенные значения могут **влиять на классификацию** и их необходимо учитывать при обучении модели.

Тогда такие значения заменяются на дополнительное уникальное числовое значение.

Таким образом можно выделить следующие способы обработки пропущенных значений для каждого качественного показателя:

1. **Удаление строк** с пропущенными значениями.
2. **Удаление столбца** с пропущенными значениями.
3. Заполнение ячеек с пропущенными значениями **новым уникальным значением**.
4. Использование методов прямого и обратного заполнения:
    a. **Метод прямого заполнения** – заполнение каждой пустой ячейки значением последней непустой ячейки из предыдущих ячеек (движение с начала в конец).
    b. **Метод обратного заполнения** – заполнение каждой пустой ячейки значением первой непустой ячейки из последующих ячеек (движение с конца в начало).

В нашем случае 31% строк только по столбцу «stalk-root» имеют пропущенные значения. Рассмотрим все 4 способа обработки пропущенных значений.

1. Удаление строк

Удалять почти треть строк из-за одного столбца нецелесообразно.

Однако, если бы доля строк с пропущенными значениями составляла несколько процентов, то удаление строк можно было бы реализовать через обновление таблицы, взяв только те ее строки, которые удовлетворяют условию:

```
таблица = таблица[условие]
```

В нашем случае необходимо взять те строки, чтобы значения определенного столбца по этим строкам не были равны символу «?»:

```
таблица = таблица[таблица[столбец] != '?']
```

То есть команда будет выглядеть следующим образом:

```
df = df[df['stalk-root'] != '?']
```

```
In [65]: df
Out[65]:
     class cap-shape cap-surface  ... spore-print-color population habitat
0        e         b           s  ...                 n          n       m
1        p         x           y  ...                 k          s       u
2        e         x           s  ...                 n          a       g
3        e         x           y  ...                 k          n       g
4        e         b           s  ...                 k          n       m
...    ...       ...         ...  ...               ...        ...     ...
7984     e         b           y  ...                 w          y       p
7999     e         x           y  ...                 w          y       p
8036     e         x           y  ...                 w          y       p
8093     p         x           y  ...                 w          c       d
8112     p         f           y  ...                 w          c       d

[5642 rows x 22 columns]
```

2. Удаление столбца

Одним из возможных решений в нашем случае является удаление столбца. Однако данный показатель может быть важным для исследователя и играть существенную роль в классификации.

Удаление столбца осуществляется с помощью метода `drop()`, либо ключевого слова `del`:

```
df = df.drop(labels = 'stalk-root', axis = 1) # Способ 1
del df['stalk-root']                          # Способ 2
```

3. Новое уникальное значение

Заполнение пустых ячеек новым уникальным значением можно осуществить следующим образом. Сначала пропущенные значения заменяются на `NaN` (через `np.nan`). Затем с помощью метода `fillna()` все значения `NaN` заменяются на 0:

```
df['stalk-root'][df['stalk-root'] == '?'] = np.nan     # Замена ? на NaN
df['stalk-root'] = df['stalk-root'].fillna(value = 0) # Замена NaN на 0
```

```
In [93]: df['stalk-root']
Out[93]:
0       c
1       e
2       e
3       c
4       c
       ..
8117    ?
8118    ?
8119    ?
8120    ?
8121    ?
Name: stalk-root, Length: 8122, dtype: object

In [94]: df['stalk-root'][df['stalk-root'] == '?'] = np.nan

In [95]: df['stalk-root']
Out[95]:
0         c
1         e
2         e
3         c
4         c
        ...
8117    NaN
8118    NaN
8119    NaN
8120    NaN
8121    NaN
Name: stalk-root, Length: 8122, dtype: object

In [96]: len(df['stalk-root'][df['stalk-root'].isna()])
Out[96]: 2480
```

```
In [98]: df['stalk-root']
Out[98]:
0       c
1       e
2       e
3       c
4       c
       ..
8117    0
8118    0
8119    0
8120    0
8121    0
Name: stalk-root, Length: 8122, dtype: object
```

Здесь можно было бы сразу символы «?» заменять на 0, однако метод `fillna()` используется при реализации прямого и обратного заполнения.

4. Метод прямого заполнения

```
df['stalk-root'][df['stalk-root'] == '?'] = np.nan           # Замена ? на NaN
df['stalk-root'] = df['stalk-root'].fillna(method = 'ffill') # Метод
```

```
In [107]: df['stalk-root']
Out[107]:
0       c
1       e
2       e
3       c
4       c
       ..
8117    c
8118    c
8119    c
8120    c
8121    c
Name: stalk-root, Length: 8122, dtype: object

In [108]: len(df['stalk-root'][df['stalk-root'].isna()])
Out[108]: 0
```

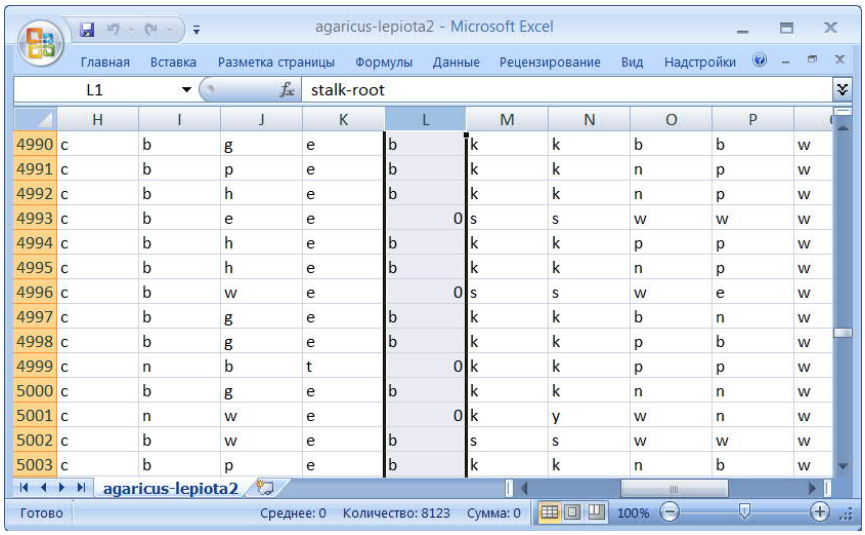
5. Метод обратного заполнения

```
df['stalk-root'][df['stalk-root'] == '?'] = np.nan           # Замена ? на NaN
df['stalk-root'] = df['stalk-root'].fillna(method = 'bfill') # Метод
```

```
In [110]: df['stalk-root']
Out[110]:
0         c
1         e
2         e
3         c
4         c
        ...
8117    NaN
8118    NaN
8119    NaN
8120    NaN
8121    NaN
Name: stalk-root, Length: 8122, dtype: object

In [111]: len(df['stalk-root'][df['stalk-root'].isna()])
Out[111]: 9
```

Выберем 3 способ обработки пропущенных значений. То есть символы «?» заменим на 0, проверим уникальные значения измененного столбца и сохраним набор данных в файле agaricus-lepiota2.csv:

```
79 df['stalk-root'][df['stalk-root'] == '?'] = np.nan     # Замена ? на NaN
80 df['stalk-root'] = df['stalk-root'].fillna(value = 0) # Замена NaN на 0
81 print(df['stalk-root'].unique())
82 df.to_csv(path_or_buf = filePath.replace('1.csv', '2.csv'),
83           sep = ';',
84           index = False)
['c' 'e' 'b' 'r' 0]
```



	H	I	J	K	L	M	N	O	P	
4990	c	b	g	e	b	k	k	b	b	w
4991	c	b	p	e	b	k	k	n	p	w
4992	c	b	h	e	b	k	k	n	p	w
4993	c	b	e	e	0	s	s	w	w	w
4994	c	b	h	e	b	k	k	p	p	w
4995	c	b	h	e	b	k	k	n	p	w
4996	c	b	w	e	0	s	s	w	e	w
4997	c	b	g	e	b	k	k	b	n	w
4998	c	b	g	e	b	k	k	p	b	w
4999	c	n	b	t	0	k	k	p	p	w
5000	c	b	g	e	b	k	k	n	n	w
5001	c	n	w	e	0	k	y	w	n	w
5002	c	b	w	e	b	s	s	w	w	w
5003	c	b	p	e	b	k	k	n	b	w

2.4 Преобразование качественных данных

Набор данных Mushroom Data Set содержит только качественные данные. Чтобы проводить анализ таких данных их требуется сначала привести в числовой вид, то есть для каждого уникального значения столбца необходимо привести в соответствие уникальное числовое значение. Например, для столбца «class» значения e (не ядовитый), p (ядовитый) нужно заменить на 0 (не ядовитый), 1 (ядовитый).

Преобразовать символьные значения качественной переменной в цифровые можно с помощью класса `sklearn.preprocessing.LabelEncoder` библиотеки `scikit-learn`.

В программном коде ниже исходные значения столбца «class» сохраняются в переменную `data_raw`, а преобразованные – в переменную `data_transform`:

```
136 from sklearn import preprocessing
137
138 le = preprocessing.LabelEncoder() # Создание экземпляра класса
139
140 # Исходные значения
141 data_raw = df['class']            # Массив исходных значений
142 print('\nИсходный массив:\n', data_raw)
143 le.fit(data_raw)                  # Заполнение экземаляра исходным массивом
144 print('Классы:', le.classes_)     # Классы (уникальные значения и. массива)
145
146 # Преобразованные значения
147 data_transform = le.transform(data_raw)       # Преобразование (преобразованный массив)
148 print('\nПреобразованный массив:\n',data_transform)
149 print('Классы:', np.unique(data_transform))   # Классы (уникальные значения п. массива)
```

```
Исходный массив:
 0       e
1       p
2       e
3       e
4       e
       ..
8117    e
8118    e
8119    e
8120    p
8121    e
Name: class, Length: 8122, dtype: object
Классы: ['e' 'p']

Преобразованный массив:
 [0 1 0 ... 0 1 0]
Классы: [0 1]
```

Таким образом, массив значений e, p был преобразован в массив значений 0, 1.

Создадим таблицу преобразованных значений `df_transform` типа DataFrame. Добавление столбца в таблицу осуществляется следующим образом:

```
таблица[новый_столбец] = массив_значений
```

```
152 # Таблица преобразованных значений
153 df_transform = pd.DataFrame()          # Пустая таблица
154 df_transform['class'] = data_transform # Добавление столбца
```

```
In [26]: df_transform
Out[26]:
      class
0         0
1         1
2         0
3         0
4         0
...     ...
8117      0
8118      0
8119      0
8120      1
8121      0

[8122 rows x 1 columns]
```

С помощью цикла преобразуем все столбцы и сохраним их в таблицу `df_transform`:

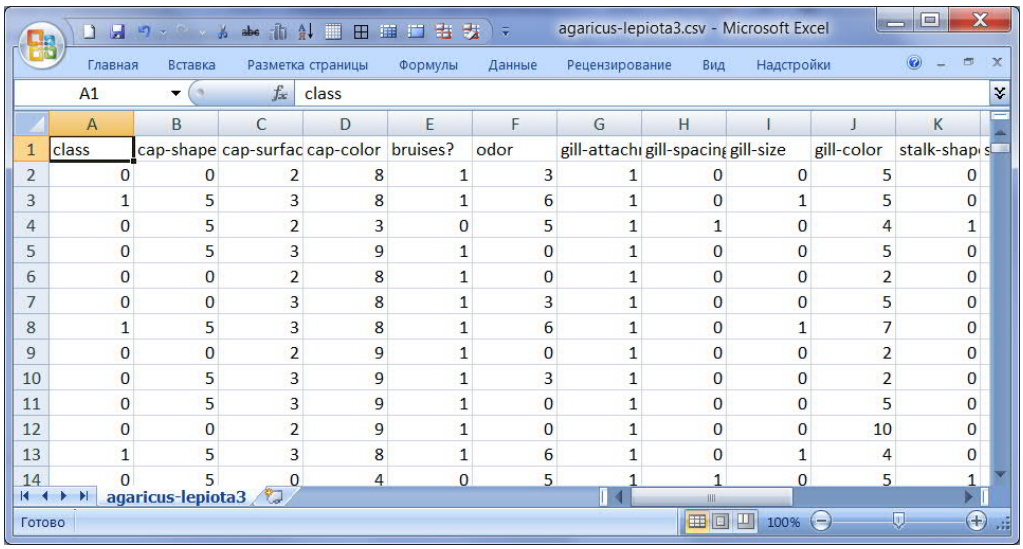
```
In [28]: df_transform
Out[28]:
      class  cap-shape  cap-surface  ...  spore-print-color  population  habitat
0         0          0            2  ...                  3           2        3
1         1          5            3  ...                  2           3        5
2         0          5            2  ...                  3           0        1
3         0          5            3  ...                  2           2        1
4         0          0            2  ...                  2           2        3
...     ...        ...          ...  ...                ...         ...      ...
8117      0          3            2  ...                  0           1        2
8118      0          5            2  ...                  0           4        2
8119      0          2            2  ...                  0           1        2
8120      1          3            3  ...                  7           4        2
8121      0          5            2  ...                  4           1        2

[8122 rows x 22 columns]
```

Для проверки правильности команд мы можем вывести уникальные значения исходных столбцов (таблица `df`) и преобразованных столбцов (таблица `df_transform`):

```
Уникальные значения столбцов:
class ['e' 'p'] [0 1]
cap-shape ['b' 'x' 's' 'f' 'k' 'c'] [0 5 4 2 3 1]
cap-surface ['s' 'y' 'f' 'g'] [2 3 0 1]
cap-color ['w' 'g' 'y' 'n' 'e' 'p' 'b' 'u' 'c' 'r'] [8 3 9 4 2 5 0 7 1 6]
bruises? ['t' 'f'] [1 0]
odor ['l' 'p' 'n' 'a' 'f' 'c' 'y' 's' 'm'] [3 6 5 0 2 1 8 7 4]
gill-attachment ['f' 'a'] [1 0]
gill-spacing ['c' 'w'] [0 1]
gill-size ['b' 'n'] [0 1]
gill-color ['n' 'k' 'g' 'p' 'w' 'h' 'u' 'e' 'b' 'r' 'y' 'o'] [ 5  4  2  7 10  3  9  1  0  8 11  6]
stalk-shape ['e' 't'] [0 1]
stalk-root ['c' 'e' 'b' 'r' 0] [2 3 1 4 0]
stalk-surface-above-ring ['s' 'f' 'k' 'y'] [2 0 1 3]
stalk-surface-below-ring ['s' 'f' 'y' 'k'] [2 0 3 1]
stalk-color-above-ring ['w' 'g' 'p' 'n' 'b' 'e' 'o' 'c' 'y'] [7 3 6 4 0 2 5 1 8]
stalk-color-below-ring ['w' 'p' 'g' 'b' 'n' 'e' 'y' 'o' 'c'] [7 6 3 0 4 2 8 5 1]
veil-color ['w' 'n' 'o' 'y'] [2 0 1 3]
ring-number ['o' 't' 'n'] [1 2 0]
ring-type ['p' 'e' 'l' 'f' 'n'] [4 0 2 1 3]
spore-print-color ['n' 'k' 'u' 'h' 'w' 'r' 'o' 'y' 'b'] [3 2 6 1 7 5 4 8 0]
population ['n' 's' 'a' 'v' 'y' 'c'] [2 3 0 4 5 1]
habitat ['m' 'u' 'g' 'd' 'p' 'w' 'l'] [3 5 1 0 4 6 2]
```

По данному списку видно соответствие уникальных буквенных значений исходных столбцов уникальным числовым значениям преобразованных столбцов.

Сохраним преобразованные данные в файл agaricus-lepiota3.csv.



	A	B	C	D	E	F	G	H	I	J	K
1	class	cap-shape	cap-surfac	cap-color	bruises?	odor	gill-attachi	gill-spacing	gill-size	gill-color	stalk-shap
2	0	0	2	8	1	3	1	0	0	5	0
3	1	5	3	8	1	6	1	0	1	5	0
4	0	5	2	3	0	5	1	1	0	4	1
5	0	5	3	9	1	0	1	0	0	5	0
6	0	0	2	8	1	0	1	0	0	2	0
7	0	0	3	8	1	3	1	0	0	5	0
8	1	5	3	8	1	6	1	0	1	7	0
9	0	0	2	9	1	0	1	0	0	2	0
10	0	5	3	9	1	3	1	0	0	2	0
11	0	5	3	9	1	0	1	0	0	5	0
12	0	0	2	9	1	0	1	0	0	10	0
13	1	5	3	8	1	6	1	0	1	4	0
14	0	5	0	4	0	5	1	1	0	5	1

2.5 Преобразование количественных данных

Преобразование количественных данных – изменение исходных данных с целью достижения необходимых условий для применения того или иного метода анализа или линейной модели (например, для стабилизации дисперсии, для приведения распределения к нормальной форме).

1. Логарифмирование

Логарифмическое преобразование как зависимых (Y), так и независимых переменных (X) часто используется на практике. При этом, экономисты зачастую работают с натуральными логарифмами, для которых используется символ ln.

$$y = \ln(x) \ , \tag{1}$$

где x – преобразуемые (исходные) данные,
y – преобразованные данные.

Применение:

- Дисперсионный анализ: для того, чтобы эффекты воздействия факторов суммировались, а не перемножались.

2. Преобразование к обратному корню

$$y = \frac{1}{\sqrt{x}} \tag{2}$$

3. Преобразование Бокса-Кокса

Преобразование Бокса-Кокса – обобщение степенных преобразований:

$$y^{(\lambda)} = \begin{cases} \dfrac{x^{\lambda} - 1}{\lambda} & \text{если } \lambda \neq 0 \\ \ln(x) & \text{если } \lambda = 0 \end{cases} , \tag{3}$$

где λ – параметр преобразования.

Преобразование Бокса-Кокса применимо только для положительных переменных. Если переменные содержат отрицательные значения, можно

либо сдвинуть распределение, добавив константу, либо использовать преобразование Йео-Джонсона.

4. Стандартизация

Стандартизация данных – процесс приведения вектора каждого признака к такому виду, что его математическое ожидание станет нулевым, а дисперсия – единичной:

$$y_i = \frac{x_i - \bar{x}}{\sigma_x} \quad , \tag{4}$$

где y_i – стандартизованное значение признака для объекта *i*,
x_i – исходное значение признака для объекта *i*,
$\bar{x}$ – среднее значение,
σ_x – стандартное отклонение.

Применение: для смещения значений признака относительно единого центра в нуле, выравнивания разброса значений.

5. Нормализация

Нормализация – процесс масштабирования вектора каждого признака, то есть приведение его к такому виду, что вектор будет иметь единичную норму (при этом есть разные способы оценки или подсчета нормы).

Применение: для приведения значений признака к одинаковому масштабу, например к диапазону [0; 1].

Виды: max-норма (5), L1-норма (6), L2-норма (7).

$$y_i = \frac{x_i}{\max(x)} \quad , \tag{5}$$

где $\max(x)$ – максимальное значение признака *x*.

$$y_i = \frac{x_i}{\|x\|_1} = \frac{x_i}{\sum_i x_i} \quad , \tag{6}$$

где $\|x\|_1$ – норма L1.

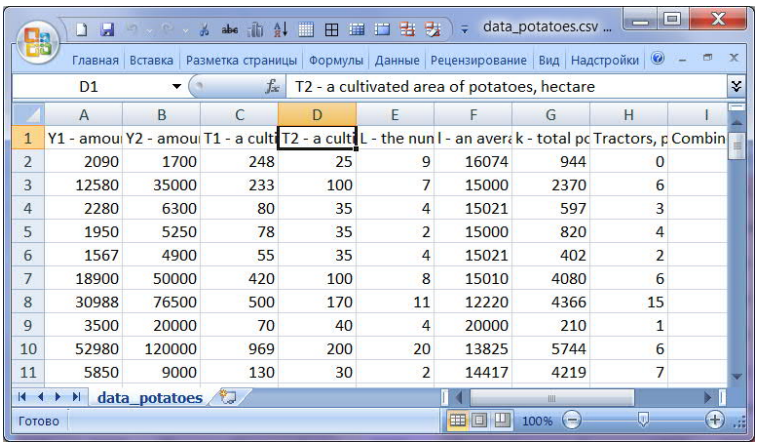
$$y_i = \frac{x_i}{\|x\|_2} = \frac{x_i}{\sqrt{\sum_i x_i^2}} \quad , \tag{7}$$

где $\|x\|_2$ – норма L2.

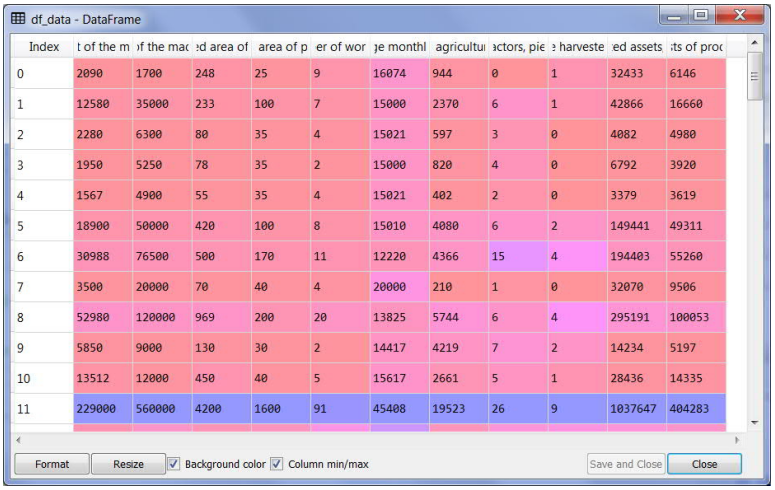
Реализация с помощью языка программирования Python

Разработаем компьютерную программу, которая будет считывать данные из файла csv, проводить преобразования данных, сохранять все результаты в сводную таблицу, строить линейные графики и гистограммы распределения по исходным и преобразованным данным.

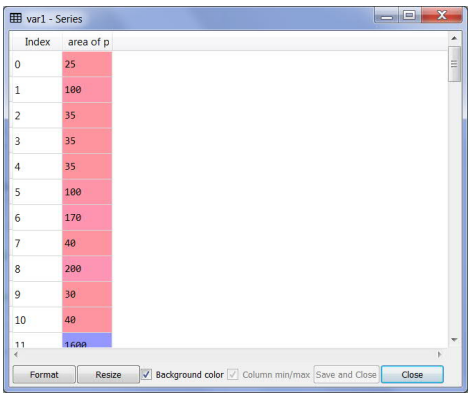
Файл с исходными данными расположен в одной директории с файлом программного кода ру и выглядит следующим образом:



С помощью функции `pandas.read_csv()` мы можем импортировать данные в среду разработки:



Выберем 4 столбец «Cultivated area of potatoes, hectares» и проведем преобразование значений данного столбца:



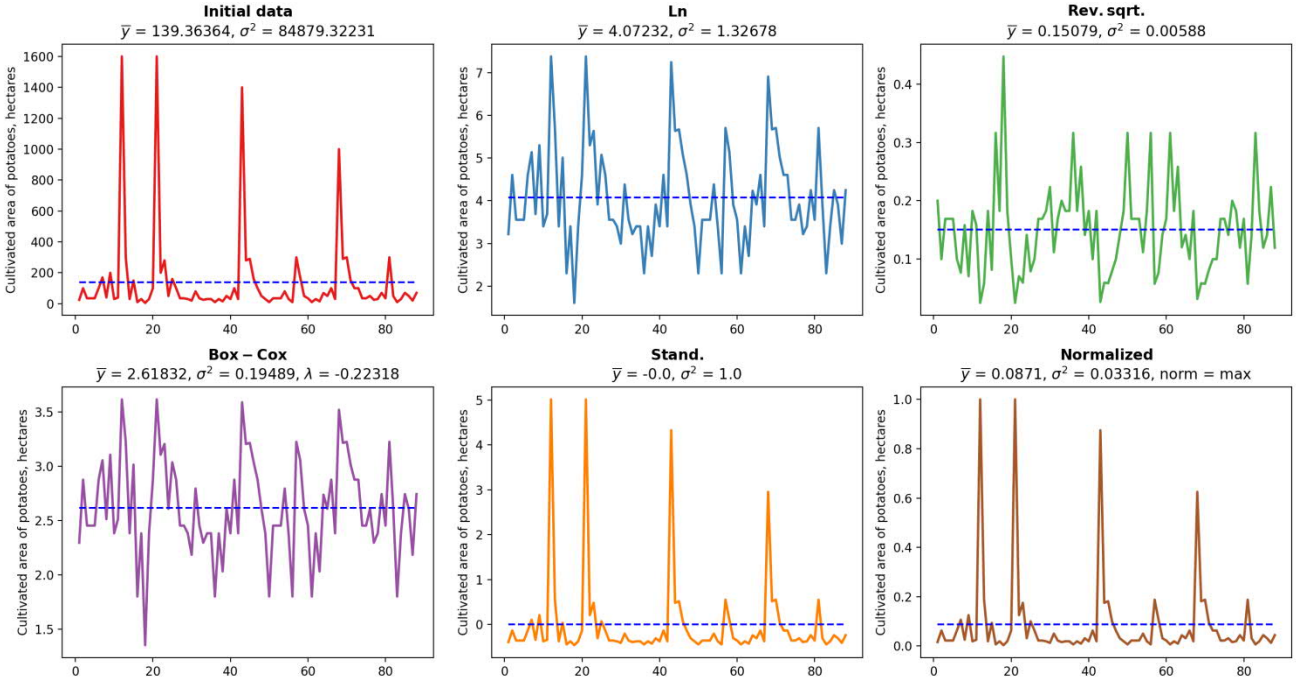
Сформируем сводную таблицу с исходными и преобразованными данными:

df_transform - DataFrame

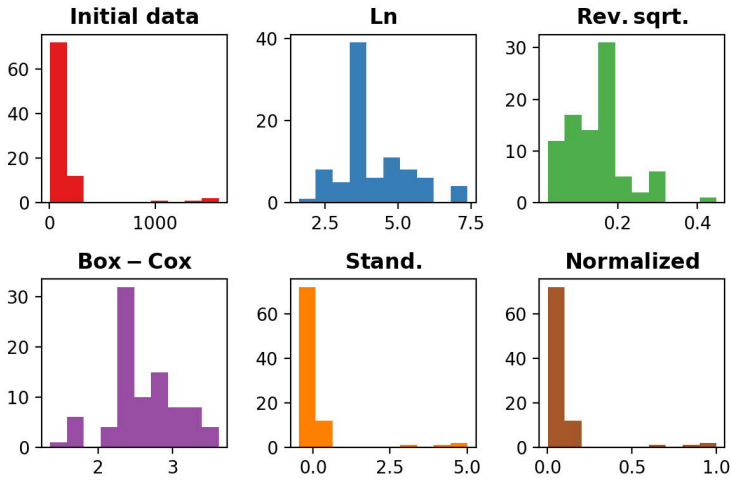
Index	Initial\ data	Ln	Rev. sqrt.	Box-Cox	Stand.	Normalized
0	25	3.21888	0.2	2.29619	-0.392543	0.015625
1	100	4.60517	0.1	2.87751	-0.135112	0.0625
2	35	3.55535	0.169031	2.45423	-0.358219	0.021875
3	35	3.55535	0.169031	2.45423	-0.358219	0.021875
4	35	3.55535	0.169031	2.45423	-0.358219	0.021875
5	100	4.60517	0.1	2.87751	-0.135112	0.0625
6	170	5.1358	0.0766965	3.05656	0.105157	0.10625
7	40	3.68888	0.158114	2.51373	-0.341057	0.025
8	200	5.29832	0.0707107	3.10729	0.208129	0.125
9	30	3.4012	0.182574	2.3833	-0.375381	0.01875
10	40	3.68888	0.158114	2.51373	-0.341057	0.025
11	1600	7.37776	0.025	3.61724	5.0135	1
12	300	5.70378	0.057735	3.22612	0.55137	0.1875

Format | Resize | Background color | Column min/max | Save and Close | Close

Построим линейные графики исходных и преобразованных данных на основе сводной таблицы. Для каждого случая рассчитаем среднее значение переменной $\bar{y}$ (синяя пунктирная линия) и дисперсию σ^2:



Построенные гистограммы говорят о том, что распределение по исходным данным не соответствует нормальному; логарифмирование и преобразование Бокса-Кокса позволило получить распределение, в определенной степени похожее на нормальное:



Примерный код программы:

```
# Подключение библиотек
import os                          # Для работы с ОС
import pandas as pd                # Таблицы DataFrame
import matplotlib.pyplot as plt    # Графики
import numpy as np                 # Log

# Настройка библиотек
#%matplotlib inline
plt.rcParams['figure.dpi'] = 200


# Пути к файлам с данными
cwd = os.getcwd()                       # Текущая директория
filePath = cwd + '\\data_potatoes.csv' # Файл CSV

# Импорт данных из файла
df_data = pd.read_csv(filePath, sep = ';')      # Таблица данных
var1 = df_data.iloc[:, 3]                       # Выбор 4 столбца

var1_name = 'Cultivated area of potatoes, hectares'     # Название для оси OY на графиках
cmap_name = 'Set1'                                      # Карта цветов (палитра) для графиков


# Функция построения линейного графика
def makePlot(Y, plot_title = '', plot_subtitle = '', x_label = '', y_label = '', color = 'blue',
             nrows = 1, ncols = 1, index = 1, plot = None):

    n = len(Y)
    X = list(range(1, n + 1, 1))   # Значения для оси OX: 1, 2, 3, …, n

    Y_mean = round(np.mean(Y), 5)  # Среднее для Y
    Y_var = round(np.var(Y), 5)    # Дисперсия для Y

    # Точечная диаграмма X, Y
    if plot is not None:
        plot1 = plot
    else:
        plot1 = plt.figure(figsize = (7, 4))             # Лист (figure) для графика

    axes1 = plot1.add_subplot(nrows, ncols, index)       # Оси для графика

    # Построение графика переменной Y
    axes1.plot(X,
               Y,
               linewidth = 2,
               marker = None,
               color = color,
               figure = plot1)

    # Построение линии средней
    axes1.plot(X,
               [Y_mean] * n,
               marker = None,
```

```
                linestyle = '--',
                color = 'blue',
                figure = plot1)

    # Настройка графика
    if plot_subtitle != '':
        plot_subtitle_addText = plot_subtitle
        plot_subtitle = r'$\overline{y}$' + f' = {Y_mean}, $\sigma^2$ = {Y_var}' + plot_subtitle_addText
    else:
        plot_subtitle = r'$\overline{y}$' + f' = {Y_mean}, $\sigma^2$ = {Y_var}'

    plot_title = r'$\bf{' + plot_title + '}$' + '\n' + plot_subtitle

    axes1.set_title(plot_title, loc = 'center')  # Название диаграммы
    axes1.set_ylabel(y_label)    # Название оси Y
    axes1.set_xlabel(x_label)    # Название оси X

    plot1.tight_layout() # Устранение наложения текста друг на друга на графике


# Функция построения множественного графика
def makeMultiPlot():

    plot1 = plt.figure(figsize = (15, 8)) # Лист (figure)

    cmap = plt.cm.get_cmap(cmap_name)     # Установка карты цветов

    # Применение функции makePlot для построения 6 линейных графиков на одном листе
    # Данные для осей OY берутся из таблицы df_transform, которая содержит 6 столбцов:
    # 1) исходные данные, 2) логарифмирование, 3) обратный корень, 4) пр-е Бокса-Кокса, 5) стандартизация,
    # 6) нормализация.
    makePlot(Y = var1, color = cmap(0.1), plot_title = 'Initial\ data', y_label = var1_name, nrows = 2,
ncols = 3, index = 1, plot = plot1)
    makePlot(Y = df_transform.iloc[:, 1], color = cmap(0.2), plot_title = 'Ln', y_label = var1_name, nrows
= 2, ncols = 3, index = 2, plot = plot1)
    makePlot(Y = df_transform.iloc[:, 2], color = cmap(0.3), plot_title = 'Rev. sqrt.', y_label =
var1_name, nrows = 2, ncols = 3, index = 3, plot = plot1)
    makePlot(Y = df_transform.iloc[:, 3], color = cmap(0.4), plot_title = 'Box-Cox', y_label = var1_name,
plot_subtitle = f', $\lambda$ = {boxcox_lambda}', nrows = 2, ncols = 3, index = 4, plot = plot1)
    makePlot(Y = df_transform.iloc[:, 4], color = cmap(0.5), plot_title = 'Stand.', y_label = var1_name,
nrows = 2, ncols = 3, index = 5, plot = plot1)
    makePlot(Y = df_transform.iloc[:, 5], color = cmap(0.7), plot_title = 'Normalized', y_label =
var1_name, plot_subtitle = ', norm = max', nrows = 2, ncols = 3, index = 6, plot = plot1)

    plot1.tight_layout()   # Устранение наложения текста друг на друга на графике


# Функция построения множественного графика гистограмм
# https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
def makeMultiHist(df):

    Y = df.iloc[:, 0]

    cmap = plt.cm.get_cmap(cmap_name)
    cmap_values = [0.1, 0.2, 0.3, 0.4, 0.5, 0.7]

    figure1, axes1 = plt.subplots(nrows = 2, ncols = 3)

    idx_counter = 0
    for row in axes1:
        for plot1 in row:
            Y = df.iloc[:, idx_counter]
            plot1.hist(Y,
                       color = cmap(cmap_values[idx_counter]))
            plot_title = r'$\bf{' + df.columns[idx_counter] + '}$'

            plot1.set_title(plot_title, loc = 'center')

            idx_counter += 1
    figure1.tight_layout()
    plt.show()


#
# Преобразование данных
#
```

```
# Таблица с исходными и преобразованными данными (изначально состоит из 1 столбца исходных данных)
df_transform = pd.DataFrame({'Initial\ data' : var1})

# 1. Логарифмирование
df_transform['Ln'] = np.log(var1) # Добавление столбца 'Ln'

# 2. Обратный корень
df_transform['Rev. sqrt.'] = 1 / np.sqrt(var1)

# 3. Преобразование Бокса-Кокса
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.boxcox.html
from scipy.stats import boxcox
boxcox_res = boxcox(var1, lmbda = None)
df_transform['Box-Cox'] = boxcox_res[0]
boxcox_lambda = round(boxcox_res[1], 5)

# 4. Стандартизация
# https://scikit-learn.org/stable/modules/preprocessing.html
from sklearn import preprocessing
var1_array = np.array(var1).reshape(-1, 1)
scaler = preprocessing.StandardScaler().fit(var1_array)
df_transform['Stand.'] = scaler.transform(var1_array)

# 5. Нормализация
df_transform['Normalized'] = preprocessing.normalize(var1_array, axis = 0, norm = 'max')


# Построение графиков с помощью вызова функций makeMultiPlot(),makeMultiHist()
makeMultiPlot()                   # Линейные графики
makeMultiHist(df = df_transform)  # Гистограммы распределения
```

3. Разведочный анализ

Разведочный анализ может проводится как до преобразования количественных данных, так и после преобразования, в зависимости от задач исследования, при этом его результаты могут существенно отличаться в этих двух случаях (до преобразования и после).

Разведочный анализ можно разделить на 2 части: визуализация данных, расчет описательных статистик.

3.1 Визуализация данных

Визуализация данных подразумевает построение различных графиков:

- **График временного ряда** (Time series graph) – стандартный график, на котором обычно по оси OY откладывают значения переменной, по оси OX – моменты времени.
- **Гистограмма** (Histogram) – график, основанный на таблице из 2 столбцов. Первый столбец – интервалы (class intervals, bins) значений переменной, делящие объекты на группы. Второй столбец – частоты (frequency), показывающие, сколько объектов входит в каждый интервал. По оси OY откладывают частоты, по оси OX – интервалы.
- **Диаграмма рассеяния** (точечная диаграмма) (XY-plot, Scatter diagram) – график, показывающий связь, отношение между двумя переменными – X и Y. Обычно Y – зависимая переменная, X – независимая, факторная переменная, влияющая на Y. По оси OY

откладывают значения переменной Y, по оси OX – значения переменной X.

Построить диаграммы рассеяния между каждой парой переменных в наборе данных `df` и гистограммы по каждой переменной позволяет библиотека `seaborn` и функция `pairplot()`:

```
import seaborn as sns
sns.pairplot(df)
```

3.2 Описательная статистика

Описательная статистика (Descriptive statistics) – обобщение свойств исследуемой переменной путем расчета статистических показателей:

- **Среднее** (Mean) – середина распределения значений переменной.

$$\bar{Y} = \frac{\sum_{i=1}^{N} Y_i}{N-1},$$

где N – размер выборки (sample size), число наблюдений;

$N-1$ – используется вместо N для расчета средней в генеральной совокупности по данным выборочной совокупности. Если размер выборки велик, то $N-1$ можно заменить на N, так как это не окажет существенного эффекта на результат;

Σ – оператор суммирования.

- **Дисперсия** (Variance, – изменчивость, колеблемость) – мера изменчивости переменной. Квадрат среднего отклонения каждого значения переменной от ее среднего значения.

$$Var = \frac{\sum_{i=1}^{N}(Y_i - \bar{Y})^2}{N-1}$$

- **Стандартное отклонение** (Standard deviation) – мера изменчивости переменной. Корень из дисперсии. Показывает на сколько в среднем отклоняются значения переменной от ее средней величины.

$$s = \sqrt{Var} = \sqrt{\frac{\sum_{i=1}^{N}(Y_i - \bar{Y})^2}{N-1}}$$

Описательную статистику для каждой переменной в наборе данных `df` (объект `DataFrame`) можно рассчитать с помощью метода `describe()`:

```
df.describe()
```

Требуется:

Подготовить набор исходных данных Mushroom Data Set для проведения классификации и представить его в файле csv.

1. Представить исходные данные в виде файла csv, добавив названия столбцов.

2. Провести предобработку данных:

- найти и удалить столбцы с одним уникальным значением;

- реализовать один из способов обработки пропущенных значений;
- преобразовать качественные данные в количественные.

3. Рассчитать описательную статистику.

4. Построить точечные диаграммы и гистограммы по переменным.

5. Привести все значения в наборе данных к диапазону [0; 1] и повторить пункты 3, 4.