Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»

Отчет по лабораторной работе № 3

«Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных»

По курсу

«Методы машинного обучения»

Выполнила:
Шаххуд Ф.М.
Студентка группы ИУ5-22М

Москва, 2020

набор данных

Департамент здравоохранения разработал отчет о проверке и систему оценки. После пров
объекта Инспектор здравоохранения рассчитывает балл на основе наблюдаемых нарушен
могут относиться к: категории высокого риска: регистрируются конкретные нарушения, ко
непосредственно связаны с передачей болезней пищевого происхождения, фальсификаци
продуктов и загрязнением поверхностей, контактирующих с пищевыми продуктами. Катего
риска: регистрирует конкретные нарушения, которые имеют умеренный риск в категорию
здравоохранения и безопасности. Низкая степень риска: регистрирует нарушения, которые
уровень риска или не имеют непосредственного риска для здоровья и безопасности насел
карточка, которая будет выдана инспектором, хранится в продовольственном учреждении
публика в этом наборе данных.Эта информация будет частью построения модели машинно
которая будет предсказывать, к какой категории относится каждое учреждение. Давайте с
на наш набор данных и увидим некоторые его строки.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Futur
  import pandas.util.testing as tm
```

```
data = pd.read_csv('/content/drive/My Drive/restaurant-scores-lives-standard.csv',
```

```
data.head()
```

	business_id	business_name	business_address	business_city	business_state
0	69618	Fancy Wheatfield Bakery	1362 Stockton St	San Francisco	CA
1	97975	BREADBELLY	1408 Clement St	San Francisco	CA
2	69487	Hakkasan San Francisco	1 Kearny St	San Francisco	CA
3	91044	Chopsticks Restaurant	4615 Mission St	San Francisco	CA
4	85987	Tselogs	552 Jones St	San Francisco	CA

```
data.shape
```

```
(53973, 17)
```

```
data.dtypes
```

```
business_id               int64
business_name            object
business_address         object
business_city            object
business_state           object
business_postal_code     object
business_latitude       float64
business_longitude      float64
business_location        object
business_phone_number   float64
inspection_id            object
inspection_date          object
inspection_score        float64
inspection_type          object
violation_id             object
violation_description    object
risk_category            object
dtype: object
```

1. Обработка пропусков в данных

1.1. Обработка пропусков в числовых данных

```
data.isnull().sum()
```

```
business_id                 0
business_name               0
business_address            0
business_city               0
business_state              0
business_postal_code     1083
business_latitude       24095
business_longitude      24095
business_location       24095
business_phone_number   36539
inspection_id               0
inspection_date             0
inspection_score        14114
inspection_type             0
violation_id            13462
violation_description   13462
risk_category           13462
dtype: int64
```

```
total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 53973
```

```
data_new_1 = data.dropna(axis=1, how='any')
(data.shape, data_new_1.shape)
```

```
((53973, 17), (53973, 8))
```

```
data_new_2 = data.dropna(axis=0, how='any')
(data.shape, data_new_2.shape)
```

```
((53973, 17), (5711, 17))
```

Если мы удалили столбцы, содержащие нулевое значение, мы получим 8 вместо 17.

Если мы удалили строку, содержащую нулевые значения, мы получим 5711 вместо 53973 с

Процент пустых значений в девяти столбцах:

```
a.columns:
 пустых значений
count = data[data[col].isnull()].shape[0]
ata[col].dtype)
ll_count>0:
ls.append(col)
erc = round((temp_null_count / total_count) * 100.0, 2)
'Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(col, dt, temp_nul
```

```
Колонка business_postal_code. Тип данных object. Количество пустых значений 1083, 2.
Колонка business_latitude. Тип данных float64. Количество пустых значений 24095, 44.
Колонка business_longitude. Тип данных float64. Количество пустых значений 24095, 44
Колонка business_location. Тип данных object. Количество пустых значений 24095, 44.6
Колонка business_phone_number. Тип данных float64. Количество пустых значений 36539,
Колонка inspection_score. Тип данных float64. Количество пустых значений 14114, 26.1
Колонка violation_id. Тип данных object. Количество пустых значений 13462, 24.94%.
Колонка violation_description. Тип данных object. Количество пустых значений 13462,
Колонка risk_category. Тип данных object. Количество пустых значений 13462, 24.94%.
```

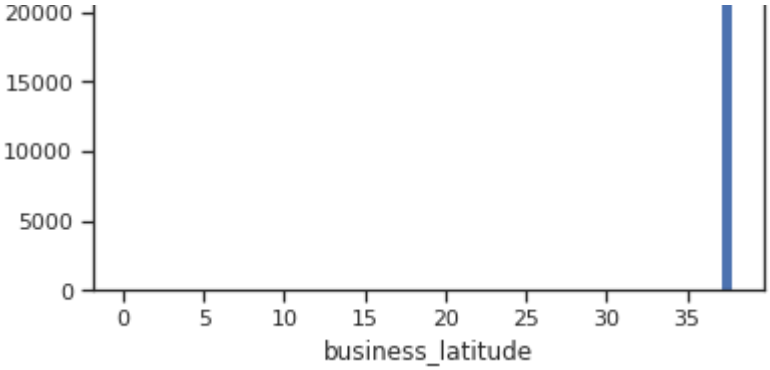
```
data_num = data[num_cols]
data_num
```

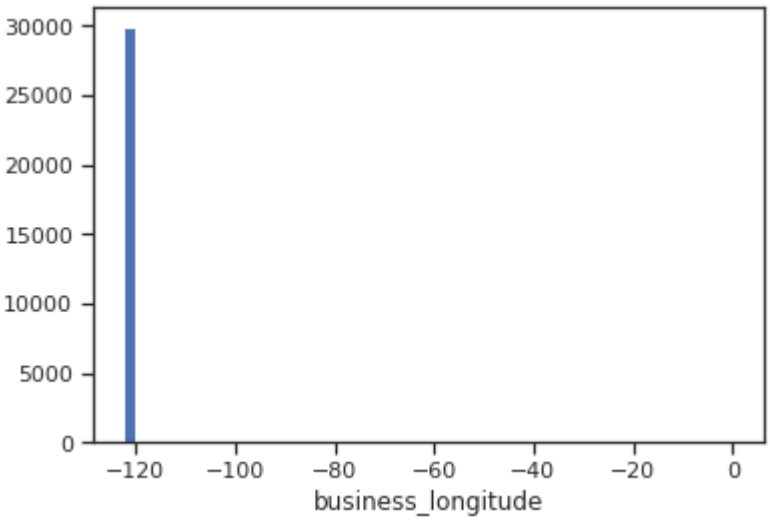
	business_postal_code	business_latitude	business_longitude	business_lo
0	94133	NaN	NaN	
1	94118	NaN	NaN	
2	94108	NaN	NaN	
3	94112	NaN	NaN	
4	94102	NaN	NaN	
...	...	...	...	
53968	94110	NaN	NaN	
53969	94134	NaN	NaN	
53970	94103	NaN	NaN	
53971	94110	NaN	NaN	
53972	94122	NaN	NaN	

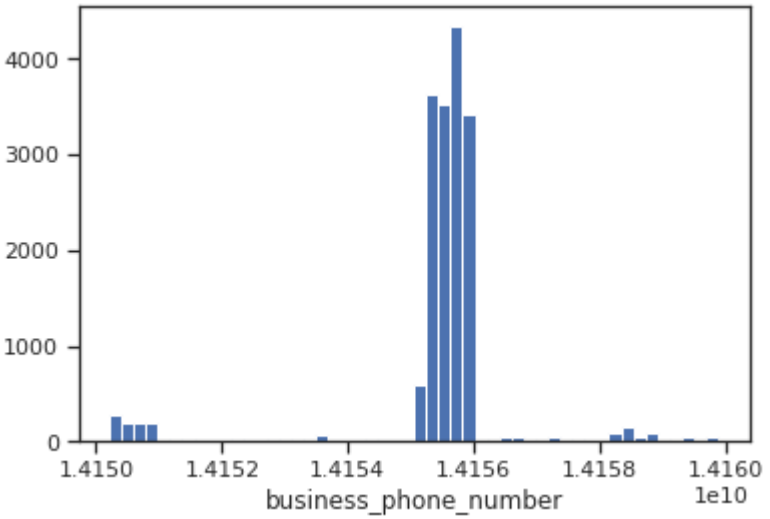
53973 rows × 9 columns

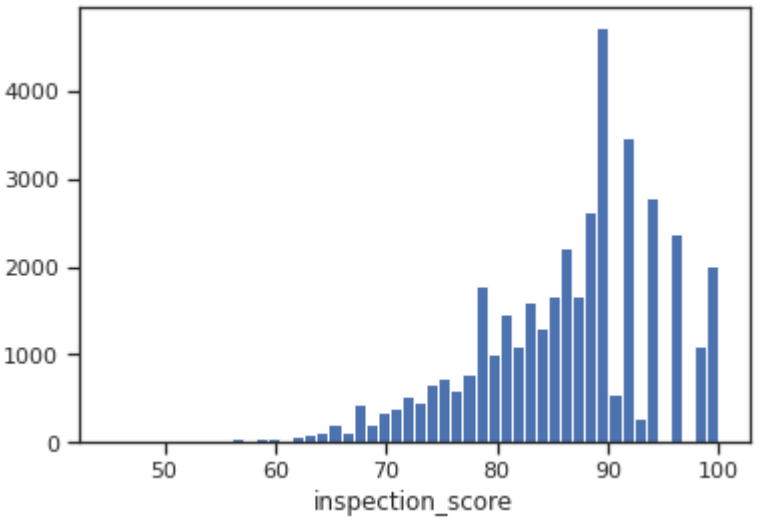
гистограмма для числовых данных

```
for col in data_num:
    if(data[col].dtype!='O'):
      plt.hist(data[col],50)
      plt.xlabel(col)
      plt.show()
```


20000
15000
10000
5000
0
0
5
10
15
20
25
30
35
business_latitude


30000
25000
20000
15000
10000
5000
0
−120
−100
−80
−60
−40
−20
0
business_longitude


4000
3000
2000
1000
0
1.4150
1.4152
1.4154
1.4156
1.4158
1.4160
1e10
business_phone_number


4000
3000
2000
1000
0
50
60
70
80
90
100
inspection_score

Мы знаем, что столбец postal_code должен быть уникальным для каждого здания, и мы мо нулевые значения, поэтому мы отбросим этот столбец.

Также в столбцах (business_latitude, business_longitude, business_location) мы не можем пре значения в них, поэтому мы заполняем их нулевыми значениями. Все столбцы (postal_code business_longitude, business_location) не имеют такого значения, поскольку столбец busines содержит нулевого значения, поэтому мы можем положиться на него при анализе данных столбцов.

```
data_num=data_num.drop(columns='business_postal_code')
temp=data_num['business_latitude'].fillna(0)
temp1=data_num['business_longitude'].fillna(0)
temp2=data_num['business_location'].fillna('---')
```

```
ı']]=pd.DataFrame({'business_latitude':temp,'business_longitude':temp1,'business_lo
```

```
data_num
```

	business_latitude	business_longitude	business_location	business_phone
0	0.0	0.0	---	
1	0.0	0.0	---	1.41

```
def test_num_impute_col(dataset, column, strategy_param,fillvalue=None):
    temp_data = dataset[[column]]

    indicator = MissingIndicator()
    mask_missing_values_only = indicator.fit_transform(temp_data)

    imp_num = SimpleImputer(missing_values=np.nan,strategy=strategy_param,fill_valu
    data_num_imp = imp_num.fit_transform(temp_data)

    filled_data = data_num_imp[mask_missing_values_only]
    data_num[[column]]=data_num_imp

    return column, strategy_param, filled_data.size, filled_data[0], filled_data[fi
```

```
dic={}
for i in data['business_name'].index:
  if ~(data.loc[i,'business_phone_number'] != data.loc[i,'business_phone_number']):
    dic[data.loc[i,'business_name']]=data.loc[i,'business_phone_number']
```

53972	0.0	0.0	---	1.41

чтобы заполнить пустые значения столбца business_phone_number, мы сначала заполним е
компании, который доступен в другой строке, и, если он недоступен, мы заполним его нуле

```
for i in data['business_name'].index:
  if (data.loc[i,'business_phone_number'] != data.loc[i,'business_phone_number']):
    if data.loc[i,'business_name'] in dic:
      data_num.loc[i,'business_phone_number']=dic[data.loc[i,'business_name']]
```

```
test_num_impute_col(data, 'business_phone_number', strategy_param='constant',fillva
```

```
('business_phone_number', 'constant', 36539, 0.0, 0.0)
```

```
test_num_impute_col(data, 'inspection_score', strategy_param='mean')
```

```
('inspection_score', 'mean', 14114, 86.23525427130636, 86.23525427130636)
```

1.2. Обработка пропусков в категориальных данных.

Мы заполняем нулевые значения в столбце protect_id константным значением Not_specifie
можем предсказать идентификатор. Мы заполняем пустые значения в столбцах (protect_d
risk_category) наиболее частыми значениями, предполагая, что такая ситуация встречается

```
test_num_impute_col(data, 'violation_id', strategy_param='constant',fillvalue='Not_
```

```
test_num_impute_col(data, 'violation_description', strategy_param='most_frequent')
test_num_impute_col(data, 'risk_category', strategy_param='most_frequent')
```

```
('risk_category', 'most_frequent', 13462, 'Low Risk', 'Low Risk')
```

Теперь мы анализируем данные, чтобы все столбцы не имели нулевых значений.

```
data_num
```

	business_latitude	business_longitude	business_location	business_phone
0	0.0	0.0	---	0.00
1	0.0	0.0	---	1.41
2	0.0	0.0	---	0.00
3	0.0	0.0	---	0.00
4	0.0	0.0	---	0.00
...	...	...	...	
53968	0.0	0.0	---	1.41
53969	0.0	0.0	---	0.00
53970	0.0	0.0	---	1.41
53971	0.0	0.0	---	1.41
53972	0.0	0.0	---	1.41

53973 rows × 8 columns

```
data_num.isnull().sum()
```

```
business_latitude        0
business_longitude       0
business_location        0
business_phone_number    0
inspection_score         0
violation_id             0
violation_description    0
risk_category            0
dtype: int64
```

2. Преобразование категориальных признаков в числовые

В нашем наборе данных у нас есть 11 атрибутов типа объекта, которые необходимо преобр значения.

```
for column in data:
  if (data[column].dtype=='O'):
    print(column,len(data[column].unique()))
```

```
business_name 5743
business_address 5646
business_city 1
business_state 1
business_postal_code 63
business_location 2369
inspection_id 24474
inspection_date 807
inspection_type 14
violation_id 35842
violation_description 66
risk_category 4
```

Мы будем использовать LabelEncoder со столбцами, которые имеют много значений, таких business_address, business_location, inspection_id, violation_id.

С business_city, business_state, inspection_date, inspection_type, violation_description, risk_categ имеют много значений, мы будем использовать Кодирование категорий наборами бинарны encoding.

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder


le = LabelEncoder()
name_le = le.fit_transform(data['business_name'])
np.unique(name_le)
```

```
array([   0,    1,    2, ..., 5740, 5741, 5742])
```

```
address_le = le.fit_transform(data['business_address'])
np.unique(address_le)
```

```
array([   0,    1,    2, ..., 5643, 5644, 5645])
```

```
location_le = le.fit_transform(data_num['business_location'])
np.unique(location_le)
```

```
array([   0,    1,    2, ..., 2366, 2367, 2368])
```

```
inspection_le = le.fit_transform(data['inspection_id'])
np.unique(inspection_le)
```

```
array([    0,     1,     2, ..., 24471, 24472, 24473])
```

```
violation_le = le.fit_transform(data_num['violation_id'])
np.unique(violation_le)
```

```
array([    0,     1,     2, ..., 35839, 35840, 35841])
```

```
ohe = OneHotEncoder()
city_ohe = ohe.fit_transform(data[['business_city']])
print(city_ohe.shape)
city_ohe.toarray()[0:3]
```

```
(53973, 1)
array([[1.],
       [1.],
       [1.]])
```

```
state_ohe = ohe.fit_transform(data[['business_state']])
print(state_ohe.shape)
state_ohe.toarray()[0:3]
```

```
(53973, 1)
array([[1.],
       [1.],
       [1.]])
```

```
inspection_date_ohe = ohe.fit_transform(data[['inspection_date']])
print(inspection_date_ohe.shape)
inspection_date_ohe.toarray()[0:3]
```

```
(53973, 807)
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
inspection_type_ohe = ohe.fit_transform(data[['inspection_type']])
print(inspection_type_ohe.shape)
inspection_type_ohe.toarray()[0:3]
```

```
(53973, 14)
array([[0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]])
```

```
violation_description_ohe = ohe.fit_transform(data_num[['violation_description']])
print(violation_description_ohe.shape)
violation_description_ohe.toarray()[0:3]
```

```
(53973, 65)
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
```

```
risk_category_ohe = ohe.fit_transform(data_num[['risk_category']])
print(risk_category_ohe.shape)
risk_category_ohe.toarray()[0:3]
```

```
(53973, 3)
array([[0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]])
```

Теперь у нас есть все категориальные данные как числовые после того, как мы их закодир
можем применять алгоритмы машинного обучения, которые принимают только числовые

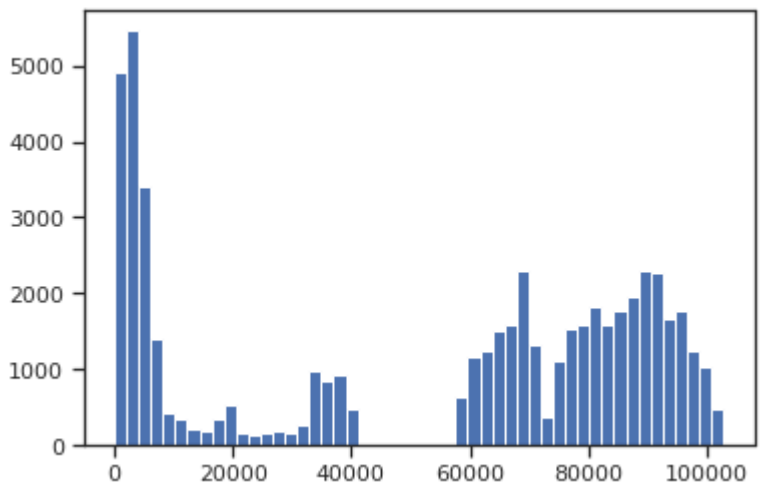
3. Масштабирование данных

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

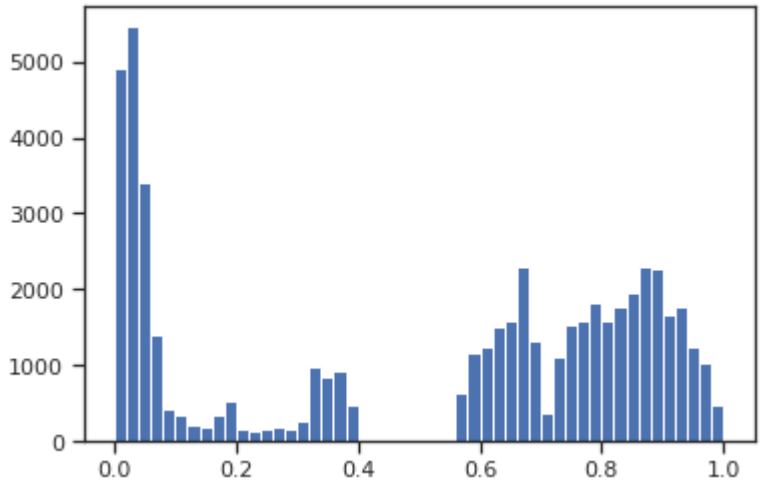
3.1. MinMax масштабирование

```
mm = MinMaxScaler()
business_id_mm = mm.fit_transform(data[['business_id']])
```

```
plt.hist(data['business_id'], 50)
plt.show()
```




```
plt.hist(business_id_mm, 50)
plt.show()
```

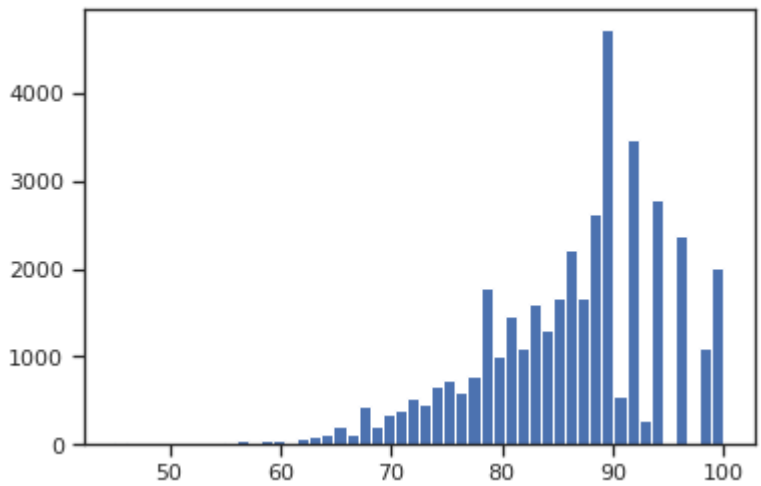


3.2. Масштабирование данных на основе Z-оценки - StandardSc

```
sc = StandardScaler()
inspection_score_sc = sc.fit_transform(data[['inspection_score']])
```

```
plt.hist(data['inspection_score'], 50)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/numpy/lib/histograms.py:839: RuntimeWar
  keep = (tmp_a >= first_edge)
/usr/local/lib/python3.6/dist-packages/numpy/lib/histograms.py:840: RuntimeWar
  keep &= (tmp_a <= last_edge)
```




```
plt.hist(inspection_score_sc, 50)
```

```
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/numpy/lib/histograms.py:839: RuntimeWar
  keep = (tmp_a >= first_edge)
/usr/local/lib/python3.6/dist-packages/numpy/lib/histograms.py:840: RuntimeWar
  keep &= (tmp_a <= last_edge)
```