In [1]:
```
import pandas as pd
import numpy as np
data = pd.read_csv('C:\\restaurant_scores_lives_standard.csv')
```

Посмотрим на то, как выглядят данные.

In [2]:
```
data.head(3)
```

Out[2]:

	business_id	business_name	business_address	business_city	business_state	business_postal_code	business_latitude	business_longitude	business_lo
0	101192	Cochinita #2	2 marina blvd Fort mason	San Francisco	CA	NaN	NaN	NaN	
1	97975	bREAdbELLY	1408 Clement St	San Francisco	CA	94118	NaN	NaN	
2	92982	Great Gold Restaurant	3161 24th St.	San Francisco	CA	94110	NaN	NaN	

3 rows × 23 columns

Размер датасета.

In [3]:
```
data.shape
```

Out[3]:
```
(53973, 23)
```
Типы колонок.

In [4]:
```
data.dtypes
```

Out[4]:
```
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
Neighborhoods (old)        float64
Police Districts           float64
Supervisor  Districts      float64
Fire Prevention Districts float64Zip
Codes                      float64
Analysis    Neighborhoods  float64
dtype: object
```

Проверим, сколько есть уникальных значений в каждой колонке, чтобы не плодить слишком много колонок после кодирования категориальных признаков One-Hot Encoding'ом.

In [5]:

```
for col in data.columns:
    print(f'В колонке {col} {data[col].nunique()} уникальных значений')
```

```
В колонке business_id 6023 уникальных значений
В колонке business_name 5572 уникальных значений
В колонке business_address 5513 уникальных значений
В колонке business_city 1 уникальных значений
В колонке business_state 1 уникальных значений
В колонке business_postal_code 61 уникальных значений
В колонке business_latitude 2291 уникальных значений
В колонке business_longitude 2320 уникальных значений
В колонке business_location 2369 уникальных значений
В колонке business_phone_number 1861 уникальных значенийВ
колонке inspection_id 21718 уникальных значений
В колонке inspection_date 800 уникальных значений
В колонке inspection_score 47 уникальных значений
В колонке inspection_type 15 уникальных значений
В колонке violation_id 31891 уникальных значений
В колонке violation_description 65 уникальных значенийВ
колонке risk_category 3 уникальных значений
В колонке Neighborhoods (old) 41 уникальных значений
В колонке Police Districts 10 уникальных значений
В колонке Supervisor Districts 11 уникальных значений
В колонке Fire Prevention Districts 15 уникальных значенийВ
колонке Zip Codes 28 уникальных значений
В колонке Analysis Neighborhoods 41 уникальных значений
```

Можем сразу дропнуть колонки business_id, business_name, business_address, business_location, inspection_id, violation_id, business_phone_number.

In [6]:

```
data = data.drop(['business_id','business_name','business_address','business_location','inspection_id','viola
```

Проверим наличие пропусков.

In [7]:

```
for col in data.columns:
    na_count = data[col].isnull().sum()
    if na_count > 0:
        print(f'В колонке {col} {na_count} пропусков = {round(100 * na_count / data.shape[0], 2)}%')
```

```
В колонке business_postal_code 1018 пропусков = 1.89%
В колонке business_latitude 19556 пропусков = 36.23%
В колонке business_longitude 19556 пропусков = 36.23%
В колонке inspection_score 13610 пропусков = 25.22%
В колонке violation_description 12870 пропусков = 23.85%В
колонке risk_category 12870 пропусков = 23.85%
В колонке Neighborhoods (old) 19594 пропусков = 36.3%
В колонке Police Districts 19594 пропусков = 36.3%
В колонке Supervisor Districts 19594 пропусков = 36.3%
В колонке Fire Prevention Districts 19646 пропусков = 36.4%В
колонке Zip Codes 19576 пропусков = 36.27%
В колонке Analysis Neighborhoods 19594 пропусков = 36.3%
```

Выводы о пригодности колонок для построения модели:

Оставим все колонки.

Предобработка данных

Заполнение пропусков

In [8]:

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
for col in data.columns:
    na_count = data[col].isnull().sum()
    if na_count > 0:
        data[col] = imputer.fit_transform(data[[col]])
```

In [9]:

```
data.isnull().any()
```

Out[9]:

```
business_city             False
business_state            False
business_postal_code      False
business_latitude         False
business_longitude        False
inspection_date           False
inspection_score          False
inspection_type           False
violation_description     False
risk_category             False
Neighborhoods (old)       False
Police Districts          False
Supervisor  Districts     False
Fire Prevention Districts False
Zip Codes                 False
Analysis Neighborhoods    False
dtype: bool
```

Разделение выборки на фичи и целевой признак

Будем предсказывать переменную "категория риска".

In [10]:

```
X = data.drop('risk_category', axis=1)y
= data['risk_category']
```

Кодирование категориальных признаков

In [11]:

```
X = pd.get_dummies(X)
```

In [12]:

```
X.head()
```

Out[12]:

	business_latitude	business_longitude	inspection_score	Neighborhoods (old)	Police Districts	Supervisor Districts	Fire Prevention Districts	Zip Codes	Analysis Neighborhoods	business_city_S Franci
0	37.80824	-122.410189	90.0	19.0	1.0	10.0	2.0	28859.0	20.0	
1	37.80824	-122.410189	96.0	19.0	1.0	10.0	2.0	28859.0	20.0	
2	37.80824	-122.410189	90.0	19.0	1.0	10.0	2.0	28859.0	20.0	
3	37.80824	-122.410189	90.0	19.0	1.0	10.0	2.0	28859.0	20.0	
4	37.80824	-122.410189	90.0	19.0	1.0	10.0	2.0	28859.0	20.0	

5 rows × 952 columns

Масштабирование числовых признаков

In [13]:

```
for col in X.columns:
    if X[col].dtype == 'float64':
        print(f'В колонке {col} данные распределены от {X[col].min()} до {X[col].max()}')
```

```
В колонке business_latitude данные распределены от 0.0 до 37.824494
В колонке business_longitude данные распределены от -122.510896 до 0.0В
колонке inspection_score данные распределены от 45.0 до 100.0
В колонке Neighborhoods (old) данные распределены от 1.0 до 41.0В
колонке Police Districts данные распределены от 1.0 до 10.0
В колонке Supervisor Districts данные распределены от 1.0 до 11.0
В колонке Fire Prevention Districts данные распределены от 1.0 до 15.0В
колонке Zip Codes данные распределены от 54.0 до 29492.0
В колонке Analysis Neighborhoods данные распределены от 1.0 до 41.0
```

Как можно заметить, масштабирование действительно нужно провести.

In [14]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
for col in X.columns:
    if X[col].dtype == 'float64':
        X[col] = scaler.fit_transform(X[[col]])
```

In [15]:

```
for col in X.columns:
    if X[col].dtype == 'float64':
        print(f'В колонке {col} данные распределены от {X[col].min()} до {X[col].max()}')
```

```
В колонке business_latitude данные распределены от 0.0 до 1.0
В колонке business_longitude данные распределены от 0.0 до 1.0
В колонке inspection_score данные распределены от 0.0 до 1.0
В колонке Neighborhoods (old) данные распределены от 0.0 до 1.0000000000000002
В колонке Police Districts данные распределены от 0.0 до 1.0
В колонке Supervisor Districts данные распределены от 0.0 до 1.0
В колонке Fire Prevention Districts данные распределены от 0.0 до 1.0В
колонке Zip Codes данные распределены от 0.0 до 0.9999999999999999
В колонке Analysis Neighborhoods данные распределены от 0.0 до 1.0000000000000002
```

Разделение выборки на тестовую и обучающую

In [16]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)
```

Обучение моделей

Решающее дерево

In [17]:

```
from sklearn.metrics import accuracy_score
```

In [18]:

```
from sklearn.tree import DecisionTreeClassifier
tree_cl =DecisionTreeClassifier(max_depth=5, random_state=1)
tree_cl.fit(X_train, y_train)
tree_cl_predicted = tree_cl.predict(X_test)
```

В качестве метрики использую accuracy.

In [19]:

```
accuracy_score(y_test, tree_cl_predicted)
```

Out[19]:

```
0.7056469542018675
```

Градиентный бустинг

XGBoost

In [20]:

```
import xgboost as xgb
```

XGboost ругается, что необходимо закодировать и целевую категориальную переменную, поэтому:

In [21]:

```
from sklearn.preprocessing import LabelEncoder
```

In [22]:

```
enc = LabelEncoder()
y_train =enc.fit_transform(y_train)y_test
= enc.fit_transform(y_test)
```

In [23]:

```
D_train = xgb.DMatrix(X_train, label=y_train)
D_test = xgb.DMatrix(X_test, label=y_test)
```

In [24]:

```
params = {'max_depth':4, 'eta':0.2}


model = xgb.train(params, D_train)
xgb_predicted = model.predict(D_test)

best_predictions = np.asarray([np.argmax(line) for line in xgb_predicted])
accuracy_score(y_test, best_predictions)
```

Out[24]:

```
0.11219801393211798
```

Как можно заметить, результат значительно хуже, чем у решающего дерева.

In [ ]: