Рубежный контроль №1

Сметанкин Кирилл ИУ5-22М

Вариант 14

In [1]:

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

In [2]:

```
data = pd.read_csv('./us-education/states_all.csv', sep=",")
data.head()
```

Out[2]:

	PRIMARY_KEY	STATE	YEAR	ENROLL	TOTAL_REVENUE	FEDERAL_REVENUE	ST
0	1992_ALABAMA	ALABAMA	1992	NaN	2678885.0	304177.0	
1	1992_ALASKA	ALASKA	1992	NaN	1049591.0	106780.0	
2	1992_ARIZONA	ARIZONA	1992	NaN	3258079.0	297888.0	
3	1992_ARKANSAS	ARKANSAS	1992	NaN	1711959.0	178571.0	
4	1992_CALIFORNIA	CALIFORNIA	1992	NaN	26260025.0	2072470.0	

5 rows × 25 columns

In [3]:

```
# размер набора данных
data.shape
```

Out[3]:

```
(1918, 25)
```

In [4]:

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

```
Всего строк: 1918
```

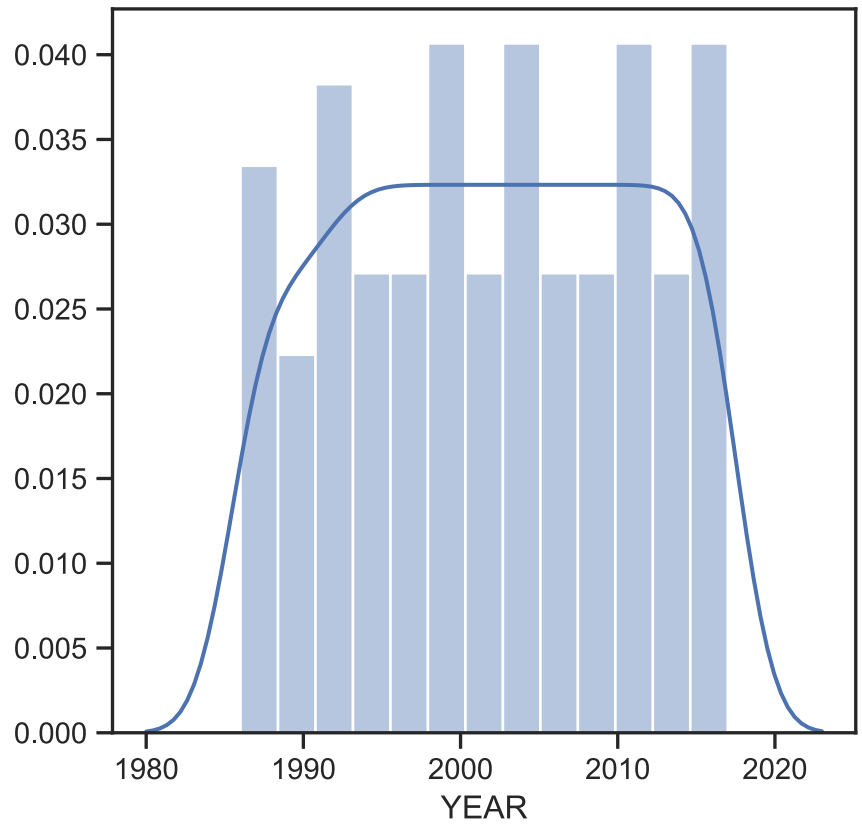
Построение гистограммы

In [5]:

```
# для колонки Year
fig, ax = plt.subplots(figsize=(5,5))
sns.distplot(data['YEAR'])
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x116802110>
```



Обработка пропусков в данных

Простые стратегии - удаление или заполнение нулями

In [6]:

```
# проверим есть ли пропущенные значения
data.isnull().sum()
```

Out[6]:

```
PRIMARY_KEY                          0
STATE                                0
YEAR                                 0
ENROLL                             694
TOTAL_REVENUE                      643
FEDERAL_REVENUE                    643
STATE_REVENUE                      643
LOCAL_REVENUE                      643
TOTAL_EXPENDITURE                  643
INSTRUCTION_EXPENDITURE            643
SUPPORT_SERVICES_EXPENDITURE       643
OTHER_EXPENDITURE                  694
CAPITAL_OUTLAY_EXPENDITURE         643
GRADES_PK_G                        376
GRADES_KG_G                        286
GRADES_4_G                         286
GRADES_8_G                         286
GRADES_12_G                        286
GRADES_1_8_G                       898
GRADES_9_12_G                      847
GRADES_ALL_G                       286
AVG_MATH_4_SCORE                  1383
AVG_MATH_8_SCORE                  1387
AVG_READING_4_SCORE               1386
AVG_READING_8_SCORE               1421
dtype: int64
```

In [7]:

```
# Удаление колонок, содержащих пустые значения
data_new_1 = data.dropna(axis=1, how='any')
(data.shape, data_new_1.shape)
```

Out[7]:

```
((1918, 25), (1918, 3))
```

In [8]:

```
# Удаление строк, содержащих пустые значения
data_new_2 = data.dropna(axis=0, how='any')
(data.shape, data_new_2.shape)
```

Out[8]:

```
((1918, 25), (306, 25))
```

In [9]:

```
# Заполнение всех пропущенных значений нулями
# В данном случае это некорректно, так как нулями заполняются в том числе категориальные ко
лонки
data_new_3 = data.fillna(0)
data_new_3.head()
```

Out[9]:

	PRIMARY_KEY	STATE	YEAR	ENROLL	TOTAL_REVENUE	FEDERAL_REVENUE	ST
0	1992_ALABAMA	ALABAMA	1992	0.0	2678885.0	304177.0	
1	1992_ALASKA	ALASKA	1992	0.0	1049591.0	106780.0	
2	1992_ARIZONA	ARIZONA	1992	0.0	3258079.0	297888.0	
3	1992_ARKANSAS	ARKANSAS	1992	0.0	1711959.0	178571.0	
4	1992_CALIFORNIA	CALIFORNIA	1992	0.0	26260025.0	2072470.0	

5 rows × 25 columns

"Внедрение значений" - импьютация (imputation)

Обработка пропусков в числовых данных

In [10]:

```
# Выберем числовые колонки с пропущенными значениями
# Цикл по колонкам датасета
num_cols = []
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='float64' or dt=='int64'):
        num_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(
col, dt, temp_null_count, temp_perc))
```

```
Колонка ENROLL. Тип данных float64. Количество пустых значений 694, 36.18%.
Колонка TOTAL_REVENUE. Тип данных float64. Количество пустых значений 643,
33.52%.
Колонка FEDERAL_REVENUE. Тип данных float64. Количество пустых значений 64
3, 33.52%.
Колонка STATE_REVENUE. Тип данных float64. Количество пустых значений 643,
33.52%.
Колонка LOCAL_REVENUE. Тип данных float64. Количество пустых значений 643,
33.52%.
Колонка TOTAL_EXPENDITURE. Тип данных float64. Количество пустых значений 6
43, 33.52%.
Колонка INSTRUCTION_EXPENDITURE. Тип данных float64. Количество пустых зна
чений 643, 33.52%.
Колонка SUPPORT_SERVICES_EXPENDITURE. Тип данных float64. Количество пуст
ых значений 643, 33.52%.
Колонка OTHER_EXPENDITURE. Тип данных float64. Количество пустых значений 6
94, 36.18%.
Колонка CAPITAL_OUTLAY_EXPENDITURE. Тип данных float64. Количество пустых
значений 643, 33.52%.
Колонка GRADES_PK_G. Тип данных float64. Количество пустых значений 376, 1
9.6%.
Колонка GRADES_KG_G. Тип данных float64. Количество пустых значений 286, 1
4.91%.
Колонка GRADES_4_G. Тип данных float64. Количество пустых значений 286, 14.
91%.
Колонка GRADES_8_G. Тип данных float64. Количество пустых значений 286, 14.
91%.
Колонка GRADES_12_G. Тип данных float64. Количество пустых значений 286, 1
4.91%.
Колонка GRADES_1_8_G. Тип данных float64. Количество пустых значений 898, 4
6.82%.
Колонка GRADES_9_12_G. Тип данных float64. Количество пустых значений 847,
44.16%.
Колонка GRADES_ALL_G. Тип данных float64. Количество пустых значений 286, 1
4.91%.
Колонка AVG_MATH_4_SCORE. Тип данных float64. Количество пустых значений 13
83, 72.11%.
Колонка AVG_MATH_8_SCORE. Тип данных float64. Количество пустых значений 13
87, 72.31%.
Колонка AVG_READING_4_SCORE. Тип данных float64. Количество пустых значений
1386, 72.26%.
Колонка AVG_READING_8_SCORE. Тип данных float64. Количество пустых значений
1421, 74.09%.
```

In [11]:

```
# Фильтр по колонкам с пропущенными значениями
data_num = data[num_cols]
data_num
```

Out[11]:

	ENROLL	TOTAL_REVENUE	FEDERAL_REVENUE	STATE_REVENUE	LOCAL_REVENUE	T
0	NaN	2678885.0	304177.0	1659028.0	715680.0	
1	NaN	1049591.0	106780.0	720711.0	222100.0	
2	NaN	3258079.0	297888.0	1369815.0	1590376.0	
3	NaN	1711959.0	178571.0	958785.0	574603.0	
4	NaN	26260025.0	2072470.0	16546514.0	7641041.0	
...	...	...	...	...	...	
1913	NaN	NaN	NaN	NaN	NaN	
1914	NaN	NaN	NaN	NaN	NaN	
1915	NaN	NaN	NaN	NaN	NaN	
1916	NaN	NaN	NaN	NaN	NaN	
1917	NaN	NaN	NaN	NaN	NaN	

1918 rows × 22 columns

In [12]:

```
# Фильтр по пустым значениям поля ENROLL
data[data['AVG_READING_8_SCORE'].isnull()]
```

Out[12]:

	PRIMARY_KEY	STATE	YEAR	ENROLL	TOTAL_REVENUE	FEDERAL_REVENUE
0	1992_ALABAMA	ALABAMA	1992	NaN	2678885.0	304177.0
4	1992_CALIFORNIA	CALIFORNIA	1992	NaN	26260025.0	2072470.0
10	1992_GEORGIA	GEORGIA	1992	NaN	5536901.0	398701.0
11	1992_HAWAII	HAWAII	1992	NaN	996809.0	71273.0
12	1992_IDAHO	IDAHO	1992	NaN	859329.0	69138.0
...	...	...	...	...	...	...
1901	1987_WYOMING	WYOMING	1987	NaN	NaN	NaN
1902	1988_WYOMING	WYOMING	1988	NaN	NaN	NaN
1903	1989_WYOMING	WYOMING	1989	NaN	NaN	NaN
1904	1990_WYOMING	WYOMING	1990	NaN	NaN	NaN
1905	1991_WYOMING	WYOMING	1991	NaN	NaN	NaN

1421 rows × 25 columns

In [13]:

```
# Запоминаем индексы строк с пустыми значениями
flt_index = data[data['AVG_READING_8_SCORE'].isnull()].index
flt_index
```

Out[13]:

```
Int64Index([   0,    4,   10,   11,   12,   13,   14,   15,   16,
17,
            ...
            1895, 1896, 1897, 1898, 1900, 1901, 1902, 1903, 1904, 19
05],
           dtype='int64', length=1421)
```

In [14]:

```
# фильтр по колонке
data_num[data_num.index.isin(flt_index)]['AVG_READING_8_SCORE']
```

Out[14]:

```
0       NaN
4       NaN
10      NaN
11      NaN
12      NaN
         ..
1901    NaN
1902    NaN
1903    NaN
1904    NaN
1905    NaN
Name: AVG_READING_8_SCORE, Length: 1421, dtype: float64
```

In [15]:

```
data_num_MasVnrArea = data_num[['AVG_READING_8_SCORE']]
data_num_MasVnrArea.head()
```

Out[15]:

	AVG_READING_8_SCORE
0	NaN
1	258.859712
2	262.169895
3	264.619665
4	NaN

In [16]:

```
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
```

In [17]:

```
# Фильтр для проверки заполнения пустых значений
indicator = MissingIndicator()
mask_missing_values_only = indicator.fit_transform(data_num_MasVnrArea)
mask_missing_values_only
```

Out[17]:

```
array([[ True],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [18]:

```
strategies=['mean', 'median','most_frequent']
```

In [19]:

```
def test_num_impute_col(dataset, column, strategy_param):
    temp_data = dataset[[column]]

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

    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(temp_data)

    filled_data = data_num_imp[mask_missing_values_only]

    return column, strategy_param, filled_data.size, filled_data[0], filled_data
[filled_data.size-1]
```

In [20]:

```
data[['AVG_READING_8_SCORE']].describe()
```

Out[20]:

	AVG_READING_8_SCORE
count	497.000000
mean	263.683325
std	6.792513
min	236.379102
25%	259.547225
50%	265.022859
75%	268.197443
max	280.499130

In [21]:

```
test_num_impute_col(data, 'AVG_READING_8_SCORE', strategies[0])
```

Out[21]:

```
('AVG_READING_8_SCORE', 'mean', 1421, 263.68332465275097, 263.683324
65275097)
```

In [22]:

```
test_num_impute_col(data, 'AVG_READING_8_SCORE', strategies[1])
```

Out[22]:

```
('AVG_READING_8_SCORE', 'median', 1421, 265.022858505901, 265.022858
505901)
```

In [23]:

```
test_num_impute_col(data, 'AVG_READING_8_SCORE', strategies[2])
```

Out[23]:

```
('AVG_READING_8_SCORE',
 'most_frequent',
 1421,
 236.37910176331403,
 236.37910176331403)
```

Обработка пропусков в категориальных данных

In [24]:

```
# Выберем категориальные колонки с пропущенными значениями (их нет)
# Цикл по колонкам датасета
cat_cols = []
for col in data.columns:
    # Количество пустых значений
    temp_null_count = data[data[col].isnull()].shape[0]
    dt = str(data[col].dtype)
    print(col, temp_null_count, dt)
    if temp_null_count>0 and (dt=='object'):
        cat_cols.append(col)
        temp_perc = round((temp_null_count / total_count) * 100.0, 2)
        print('Колонка {}. Тип данных {}. Количество пустых значений {}, {}%.'.format(
col, dt, temp_null_count, temp_perc))
```

```
PRIMARY_KEY 0 object
STATE 0 object
YEAR 0 int64
ENROLL 694 float64
TOTAL_REVENUE 643 float64
FEDERAL_REVENUE 643 float64
STATE_REVENUE 643 float64
LOCAL_REVENUE 643 float64
TOTAL_EXPENDITURE 643 float64
INSTRUCTION_EXPENDITURE 643 float64
SUPPORT_SERVICES_EXPENDITURE 643 float64
OTHER_EXPENDITURE 694 float64
CAPITAL_OUTLAY_EXPENDITURE 643 float64
GRADES_PK_G 376 float64
GRADES_KG_G 286 float64
GRADES_4_G 286 float64
GRADES_8_G 286 float64
GRADES_12_G 286 float64
GRADES_1_8_G 898 float64
GRADES_9_12_G 847 float64
GRADES_ALL_G 286 float64
AVG_MATH_4_SCORE 1383 float64
AVG_MATH_8_SCORE 1387 float64
AVG_READING_4_SCORE 1386 float64
AVG_READING_8_SCORE 1421 float64
```

In [25]:

```
cat_temp_data = data[['STATE']]
cat_temp_data.head()
```

Out[25]:

	STATE
0	ALABAMA
1	ALASKA
2	ARIZONA
3	ARKANSAS
4	CALIFORNIA

In [26]:

```
cat_temp_data['STATE'].unique()
```

Out[26]:

```
array(['ALABAMA', 'ALASKA', 'ARIZONA', 'ARKANSAS', 'CALIFORNIA',
       'COLORADO', 'CONNECTICUT', 'DELAWARE', 'DISTRICT_OF_COLUMBI
A',
       'FLORIDA', 'GEORGIA', 'HAWAII', 'IDAHO', 'ILLINOIS', 'INDIAN
A',
       'IOWA', 'KANSAS', 'KENTUCKY', 'LOUISIANA', 'MAINE', 'MARYLAN
D',
       'MASSACHUSETTS', 'MICHIGAN', 'MINNESOTA', 'MISSISSIPPI',
       'MISSOURI', 'MONTANA', 'NEBRASKA', 'NEVADA', 'NEW_HAMPSHIRE',
       'NEW_JERSEY', 'NEW_MEXICO', 'NEW_YORK', 'NORTH_CAROLINA',
       'NORTH_DAKOTA', 'OHIO', 'OKLAHOMA', 'OREGON', 'PENNSYLVANIA',
       'RHODE_ISLAND', 'SOUTH_CAROLINA', 'SOUTH_DAKOTA', 'TENNESSE
E',
       'TEXAS', 'UTAH', 'VERMONT', 'VIRGINIA', 'WASHINGTON',
       'WEST_VIRGINIA', 'WISCONSIN', 'WYOMING', 'DISTRICT OF COLUMBI
A',
       'NEW HAMPSHIRE', 'NEW JERSEY', 'NEW MEXICO', 'NEW YORK',
       'NORTH CAROLINA', 'NORTH DAKOTA', 'RHODE ISLAND', 'SOUTH CARO
LINA',
       'SOUTH DAKOTA', 'WEST VIRGINIA'], dtype=object)
```

In [27]:

```
cat_temp_data[cat_temp_data['STATE'].isnull()].shape
```

Out[27]:

```
(0, 1)
```

In [28]:

```
# Импьютация наиболее частыми значениями
imp2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data_imp2 = imp2.fit_transform(cat_temp_data)
data_imp2
```

Out[28]:

```
array([['ALABAMA'],
       ['ALASKA'],
       ['ARIZONA'],
       ...,
       ['SOUTH_CAROLINA'],
       ['SOUTH_DAKOTA'],
       ['WEST_VIRGINIA']], dtype=object)
```

In [29]:

```
# Пустые значения отсутствуют
np.unique(data_imp2)
```

Out[29]:

```
array(['ALABAMA', 'ALASKA', 'ARIZONA', 'ARKANSAS', 'CALIFORNIA',
       'COLORADO', 'CONNECTICUT', 'DELAWARE', 'DISTRICT OF COLUMBI
A',
       'DISTRICT_OF_COLUMBIA', 'FLORIDA', 'GEORGIA', 'HAWAII', 'IDAH
O',
       'ILLINOIS', 'INDIANA', 'IOWA', 'KANSAS', 'KENTUCKY', 'LOUISIA
NA',
       'MAINE', 'MARYLAND', 'MASSACHUSETTS', 'MICHIGAN', 'MINNESOT
A',
       'MISSISSIPPI', 'MISSOURI', 'MONTANA', 'NEBRASKA', 'NEVADA',
       'NEW HAMPSHIRE', 'NEW JERSEY', 'NEW MEXICO', 'NEW YORK',
       'NEW_HAMPSHIRE', 'NEW_JERSEY', 'NEW_MEXICO', 'NEW_YORK',
       'NORTH CAROLINA', 'NORTH DAKOTA', 'NORTH_CAROLINA', 'NORTH_DA
KOTA',
       'OHIO', 'OKLAHOMA', 'OREGON', 'PENNSYLVANIA', 'RHODE ISLAND',
       'RHODE_ISLAND', 'SOUTH CAROLINA', 'SOUTH DAKOTA', 'SOUTH_CARO
LINA',
       'SOUTH_DAKOTA', 'TENNESSEE', 'TEXAS', 'UTAH', 'VERMONT',
       'VIRGINIA', 'WASHINGTON', 'WEST VIRGINIA', 'WEST_VIRGINIA',
       'WISCONSIN', 'WYOMING'], dtype=object)
```

In [30]:

```
# Импьютация константой
imp3 = SimpleImputer(missing_values=np.nan, strategy='constant', fill_value=
'!!!')
data_imp3 = imp3.fit_transform(cat_temp_data)
data_imp3
```

Out[30]:

```
array([['ALABAMA'],
       ['ALASKA'],
       ['ARIZONA'],
       ...,
       ['SOUTH_CAROLINA'],
       ['SOUTH_DAKOTA'],
       ['WEST_VIRGINIA']], dtype=object)
```

In [31]:

```
np.unique(data_imp3)
```

Out[31]:

```
array(['ALABAMA', 'ALASKA', 'ARIZONA', 'ARKANSAS', 'CALIFORNIA',
       'COLORADO', 'CONNECTICUT', 'DELAWARE', 'DISTRICT OF COLUMBI
A',
       'DISTRICT_OF_COLUMBIA', 'FLORIDA', 'GEORGIA', 'HAWAII', 'IDAH
O',
       'ILLINOIS', 'INDIANA', 'IOWA', 'KANSAS', 'KENTUCKY', 'LOUISIA
NA',
       'MAINE', 'MARYLAND', 'MASSACHUSETTS', 'MICHIGAN', 'MINNESOT
A',
       'MISSISSIPPI', 'MISSOURI', 'MONTANA', 'NEBRASKA', 'NEVADA',
       'NEW HAMPSHIRE', 'NEW JERSEY', 'NEW MEXICO', 'NEW YORK',
       'NEW_HAMPSHIRE', 'NEW_JERSEY', 'NEW_MEXICO', 'NEW_YORK',
       'NORTH CAROLINA', 'NORTH DAKOTA', 'NORTH_CAROLINA', 'NORTH_DA
KOTA',
       'OHIO', 'OKLAHOMA', 'OREGON', 'PENNSYLVANIA', 'RHODE ISLAND',
       'RHODE_ISLAND', 'SOUTH CAROLINA', 'SOUTH DAKOTA', 'SOUTH_CARO
LINA',
       'SOUTH_DAKOTA', 'TENNESSEE', 'TEXAS', 'UTAH', 'VERMONT',
       'VIRGINIA', 'WASHINGTON', 'WEST VIRGINIA', 'WEST_VIRGINIA',
       'WISCONSIN', 'WYOMING'], dtype=object)
```

In [32]:

```
data_imp3[data_imp3=='!!!'].size
```

Out[32]:

```
0
```

Какие способы обработки пропусков в данных для категориальных и количественных признаков Вы использовали? -- удаление строк и колонок с пустыми значениями, заполнение всех пропущенных значений нулями

Какие признаки Вы будете использовать для дальнейшего построения моделей машинного обучения и почему? -- для дальнейшего построения моделей будем использовать категориальные признаки со стратегиями "most_frequent" или "constant" для корректной работы класса SimpleImputer

Масштабирование данных

MinMax масштабирование

In [33]:

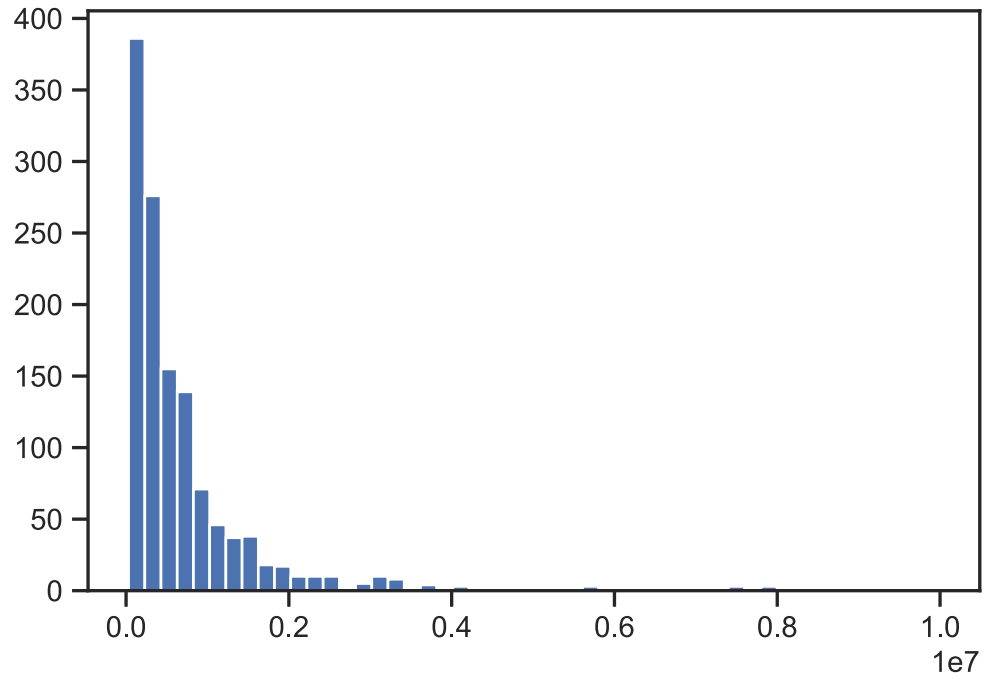
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer
```

In [34]:

```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[['FEDERAL_REVENUE']])
```

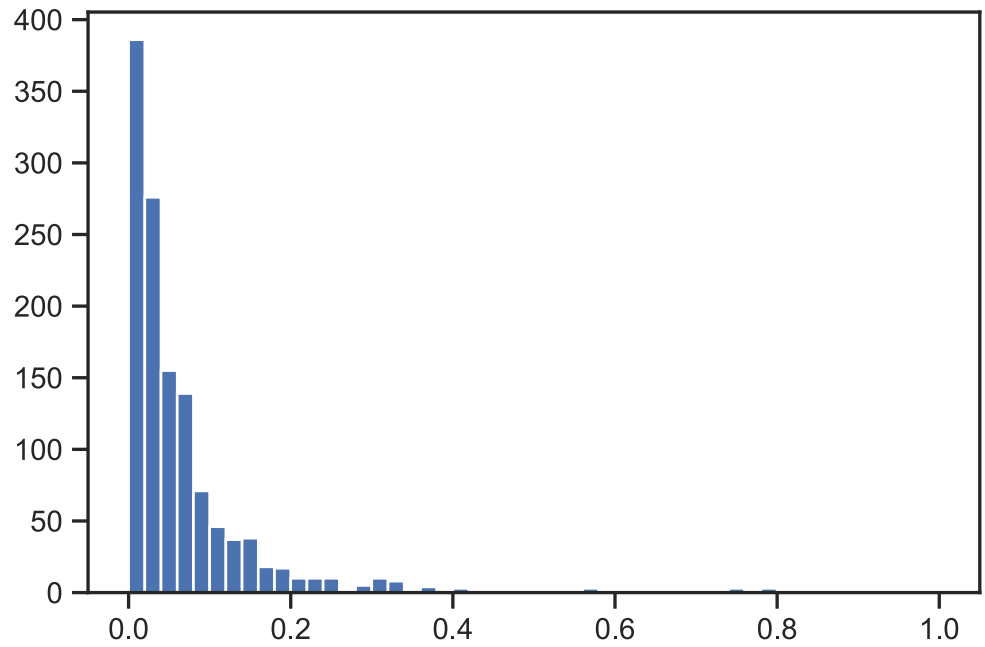
In [35]:

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



In [36]:

```
plt.hist(sc1_data, 50)
plt.show()
```



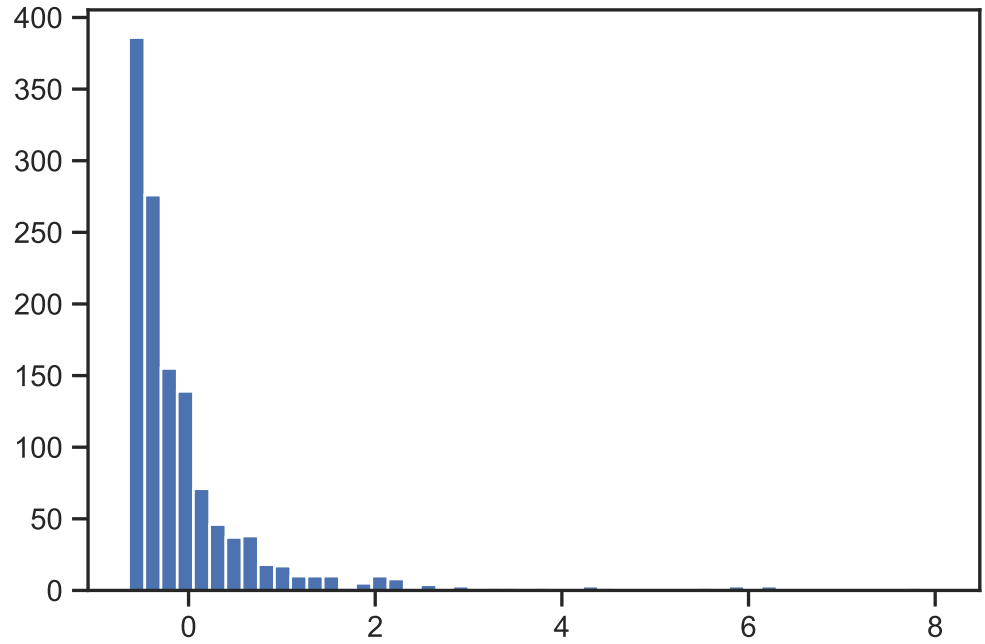
Масштабирование данных на основе Z-оценки - StandardScaler

In [37]:

```
sc2 = StandardScaler()
sc2_data = sc2.fit_transform(data[['FEDERAL_REVENUE']])
```

In [38]:

```
plt.hist(sc2_data, 50)
plt.show()
```



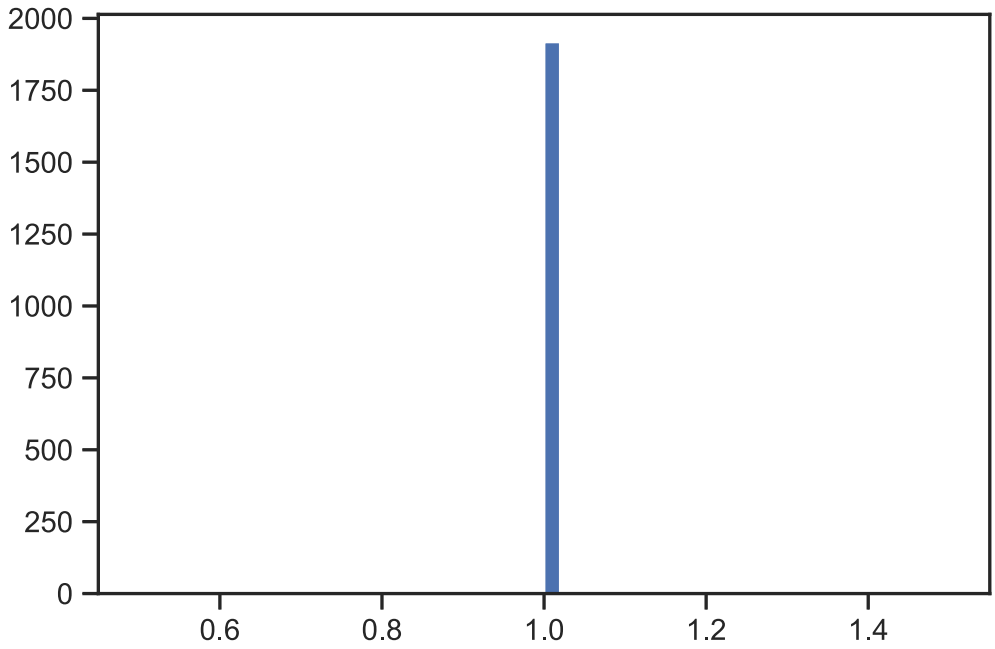
Нормализация данных

In [39]:

```
sc3 = Normalizer()
sc3_data = sc3.fit_transform(data[['YEAR']])
```

In [40]:

```
plt.hist(sc3_data, 50)
plt.show()
```



Преобразование категориальных признаков в количественные

In [41]:

```
cat_enc = pd.DataFrame({'c1':data_imp2.T[0]})
cat_enc
```

Out[41]:

	c1
0	ALABAMA
1	ALASKA
2	ARIZONA
3	ARKANSAS
4	CALIFORNIA
...	...
1913	NORTH_DAKOTA
1914	RHODE_ISLAND
1915	SOUTH_CAROLINA
1916	SOUTH_DAKOTA
1917	WEST_VIRGINIA

1918 rows × 1 columns

Кодирование категорий целочисленными значениями - label encoding

In [42]:

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

In [43]:

```
le = LabelEncoder()
cat_enc_le = le.fit_transform(cat_enc['c1'])
```

In [44]:

```
cat_enc['c1'].unique()
```

Out[44]:

```
array(['ALABAMA', 'ALASKA', 'ARIZONA', 'ARKANSAS', 'CALIFORNIA',
       'COLORADO', 'CONNECTICUT', 'DELAWARE', 'DISTRICT_OF_COLUMBI
A',
       'FLORIDA', 'GEORGIA', 'HAWAII', 'IDAHO', 'ILLINOIS', 'INDIAN
A',
       'IOWA', 'KANSAS', 'KENTUCKY', 'LOUISIANA', 'MAINE', 'MARYLAN
D',
       'MASSACHUSETTS', 'MICHIGAN', 'MINNESOTA', 'MISSISSIPPI',
       'MISSOURI', 'MONTANA', 'NEBRASKA', 'NEVADA', 'NEW_HAMPSHIRE',
       'NEW_JERSEY', 'NEW_MEXICO', 'NEW_YORK', 'NORTH_CAROLINA',
       'NORTH_DAKOTA', 'OHIO', 'OKLAHOMA', 'OREGON', 'PENNSYLVANIA',
       'RHODE_ISLAND', 'SOUTH_CAROLINA', 'SOUTH_DAKOTA', 'TENNESSE
E',
       'TEXAS', 'UTAH', 'VERMONT', 'VIRGINIA', 'WASHINGTON',
       'WEST_VIRGINIA', 'WISCONSIN', 'WYOMING', 'DISTRICT OF COLUMBI
A',
       'NEW HAMPSHIRE', 'NEW JERSEY', 'NEW MEXICO', 'NEW YORK',
       'NORTH CAROLINA', 'NORTH DAKOTA', 'RHODE ISLAND', 'SOUTH CARO
LINA',
       'SOUTH DAKOTA', 'WEST VIRGINIA'], dtype=object)
```

In [45]:

```
np.unique(cat_enc_le)
```

Out[45]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 1
5, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 3
2, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 4
9, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61])
```

In [46]:

```
le.inverse_transform([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14
, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61])
```

Out[46]:

```
array(['ALABAMA', 'ALASKA', 'ARIZONA', 'ARKANSAS', 'CALIFORNIA',
       'COLORADO', 'CONNECTICUT', 'DELAWARE', 'DISTRICT OF COLUMBI
A',
       'DISTRICT_OF_COLUMBIA', 'FLORIDA', 'GEORGIA', 'HAWAII', 'IDAH
O',
       'ILLINOIS', 'INDIANA', 'IOWA', 'KANSAS', 'KENTUCKY', 'LOUISIA
NA',
       'MAINE', 'MARYLAND', 'MASSACHUSETTS', 'MICHIGAN', 'MINNESOT
A',
       'MISSISSIPPI', 'MISSOURI', 'MONTANA', 'NEBRASKA', 'NEVADA',
       'NEW HAMPSHIRE', 'NEW JERSEY', 'NEW MEXICO', 'NEW YORK',
       'NEW_HAMPSHIRE', 'NEW_JERSEY', 'NEW_MEXICO', 'NEW_YORK',
       'NORTH CAROLINA', 'NORTH DAKOTA', 'NORTH_CAROLINA', 'NORTH_DA
KOTA',
       'OHIO', 'OKLAHOMA', 'OREGON', 'PENNSYLVANIA', 'RHODE ISLAND',
       'RHODE_ISLAND', 'SOUTH CAROLINA', 'SOUTH DAKOTA', 'SOUTH_CARO
LINA',
       'SOUTH_DAKOTA', 'TENNESSEE', 'TEXAS', 'UTAH', 'VERMONT',
       'VIRGINIA', 'WASHINGTON', 'WEST VIRGINIA', 'WEST_VIRGINIA',
       'WISCONSIN', 'WYOMING'], dtype=object)
```

Кодирование категорий наборами бинарных значений - one-hot encoding

In [47]:

```
ohe = OneHotEncoder()
cat_enc_ohe = ohe.fit_transform(cat_enc[['c1']])
```

In [48]:

```
cat_enc.shape
```

Out[48]:

```
(1918, 1)
```

In [49]:

```
cat_enc_ohe.shape
```

Out[49]:

```
(1918, 62)
```

In [50]:

```
cat_enc_ohe
```

Out[50]:

```
<1918x62 sparse matrix of type '<class 'numpy.float64'>'
        with 1918 stored elements in Compressed Sparse Row format>
```

In [51]:

```
cat_enc_ohe.todense()[0:10]
```

Out[51]:

```
matrix([[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., 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., 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., 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., 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., 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., 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., 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., 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.],
       [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.]])
```

In [52]:

```
cat_enc.head(10)
```

Out[52]:

	c1
0	ALABAMA
1	ALASKA
2	ARIZONA
3	ARKANSAS
4	CALIFORNIA
5	COLORADO
6	CONNECTICUT
7	DELAWARE
8	DISTRICT_OF_COLUMBIA
9	FLORIDA

Pandas get_dummies - быстрый вариант one-hot кодирования

In [53]:

```
pd.get_dummies(cat_enc).head()
```

Out[53]:

	c1_ALABAMA	c1_ALASKA	c1_ARIZONA	c1_ARKANSAS	c1_CALIFORNIA	c1_COLORADO
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0

5 rows × 62 columns

In [54]:

```
pd.get_dummies(cat_temp_data, dummy_na=True).head()
```

Out[54]:

	STATE_ALABAMA	STATE_ALASKA	STATE_ARIZONA	STATE_ARKANSAS	STATE_CALIFORNIA
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	0
4	0	0	0	0	1

5 rows × 63 columns

In [ ]:

```

```