РК №2

Овчинников Данила Алексеевич ИУ5-62Б

Вариант 14

Задание. Для заданного набора данных (вариант 14) постройте модели классификации. Для построения моделей используйте методы опорных векторов и случайный лес. Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д. Данные: https://www.kaggle.com/noriuk/us-education-datasets-unification-project (файл states_all.csv)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import seaborn as sns
import scipy
import plotly
import missingno as msno
from numpy import nan
from sklearn.impute import SimpleImputer, MissingIndicator
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, median_absolut
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
```

Загрузим датасет и выведем информацию о нем.

In [2]:
```
dataset = pd.read_csv('states_all.csv')
```

In [3]:
```
dataset.head(5)
```

Out[3]:

	PRIMARY_KEY	STATE	YEAR	ENROLL	TOTAL_REVENUE	FEDERAL_REVENUE	STATE_REV
0	1992_ALABAMA	ALABAMA	1992	NaN	2678885.0	304177.0	165
1	1992_ALASKA	ALASKA	1992	NaN	1049591.0	106780.0	72
2	1992_ARIZONA	ARIZONA	1992	NaN	3258079.0	297888.0	136
3	1992_ARKANSAS	ARKANSAS	1992	NaN	1711959.0	178571.0	95
4	1992_CALIFORNIA	CALIFORNIA	1992	NaN	26260025.0	2072470.0	1654

5 rows × 25 columns

In [4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1715 entries, 0 to 1714
Data columns (total 25 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   PRIMARY_KEY                    1715 non-null   object 
 1   STATE                          1715 non-null   object 
 2   YEAR                           1715 non-null   int64  
 3   ENROLL                         1224 non-null   float64
 4   TOTAL_REVENUE                  1275 non-null   float64
 5   FEDERAL_REVENUE                1275 non-null   float64
 6   STATE_REVENUE                  1275 non-null   float64
 7   LOCAL_REVENUE                  1275 non-null   float64
 8   TOTAL_EXPENDITURE              1275 non-null   float64
 9   INSTRUCTION_EXPENDITURE        1275 non-null   float64
 10  SUPPORT_SERVICES_EXPENDITURE   1275 non-null   float64
 11  OTHER_EXPENDITURE              1224 non-null   float64
 12  CAPITAL_OUTLAY_EXPENDITURE     1275 non-null   float64
 13  GRADES_PK_G                    1542 non-null   float64
 14  GRADES_KG_G                    1632 non-null   float64
 15  GRADES_4_G                     1632 non-null   float64
 16  GRADES_8_G                     1632 non-null   float64
 17  GRADES_12_G                    1632 non-null   float64
 18  GRADES_1_8_G                   1020 non-null   float64
 19  GRADES_9_12_G                  1071 non-null   float64
 20  GRADES_ALL_G                   1632 non-null   float64
 21  AVG_MATH_4_SCORE               565 non-null    float64
 22  AVG_MATH_8_SCORE               602 non-null    float64
 23  AVG_READING_4_SCORE            650 non-null    float64
 24  AVG_READING_8_SCORE            562 non-null    float64
dtypes: float64(22), int64(1), object(2)
memory usage: 335.1+ KB
```

Подсчитаем количество и процент пропусков по столбцам.

In [5]:
```
for col in dataset.columns:
    pct_missing = np.mean(dataset[col].isnull())
    print('{}: {} - {}%'.format(col, dataset[col].isna().sum(), round(pct_missing*1
```

```
PRIMARY_KEY: 0 - 0.0%
STATE: 0 - 0.0%
YEAR: 0 - 0.0%
ENROLL: 491 - 28.63%
TOTAL_REVENUE: 440 - 25.66%
FEDERAL_REVENUE: 440 - 25.66%
STATE_REVENUE: 440 - 25.66%
LOCAL_REVENUE: 440 - 25.66%
TOTAL_EXPENDITURE: 440 - 25.66%
INSTRUCTION_EXPENDITURE: 440 - 25.66%
SUPPORT_SERVICES_EXPENDITURE: 440 - 25.66%
OTHER_EXPENDITURE: 491 - 28.63%
CAPITAL_OUTLAY_EXPENDITURE: 440 - 25.66%
GRADES_PK_G: 173 - 10.09%
GRADES_KG_G: 83 - 4.84%
GRADES_4_G: 83 - 4.84%
GRADES_8_G: 83 - 4.84%
GRADES_12_G: 83 - 4.84%
GRADES_1_8_G: 695 - 40.52%
GRADES_9_12_G: 644 - 37.55%
GRADES_ALL_G: 83 - 4.84%
AVG_MATH_4_SCORE: 1150 - 67.06%
AVG_MATH_8_SCORE: 1113 - 64.9%
AVG_READING_4_SCORE: 1065 - 62.1%
AVG_READING_8_SCORE: 1153 - 67.23%
```

Обработка пропусков.

Последние 4 столбца невозможно восстановить из-за слишком большого процента пропусков. Посмотрим на корреляционную матрицу признаков. Также удалим столбец PRIMARY_KEY, так как он является первичным ключом и не нужен для построения модели.

In [6]:
```
dataset.drop(['PRIMARY_KEY'], axis=1, inplace=True)
```

Видно, что последние 4 признака не коррелируют с остальными. Так как мы не будем выбирать эти признаки в качестве целевых, их можно вырезать из датасета и не использовать для построения модели.

In [7]:
```
dataset.drop(['AVG_MATH_4_SCORE',
              'AVG_MATH_8_SCORE',
              'AVG_READING_4_SCORE',
              'AVG_READING_8_SCORE'], axis=1, inplace=True)
```

В строках с процентом пропусков >20 заполнение приведет к резкому снижению достоверности. Условия задачи позволяют сократить набор данных, поэтому лучшим решением будет удалить строки с пропусками.

По количеству пропусков очевидно, что в столбцах ENROLL, TOTAL_REVENUE, ... CAPITAL_OUTLAY_EXPENDITURE отсутствующие значения находятся на одних и тех же строчках, поэтому достаточно очистить один из этих столбцов.

Также почистим пропуски в паре столбцов GRADES_1_8_G и GRADES_9_12_G

In [8]:
```
dataset.dropna(subset=['ENROLL'], axis=0, inplace=True)
dataset.dropna(subset=['GRADES_1_8_G', 'GRADES_9_12_G'], axis=0, inplace=True)
```

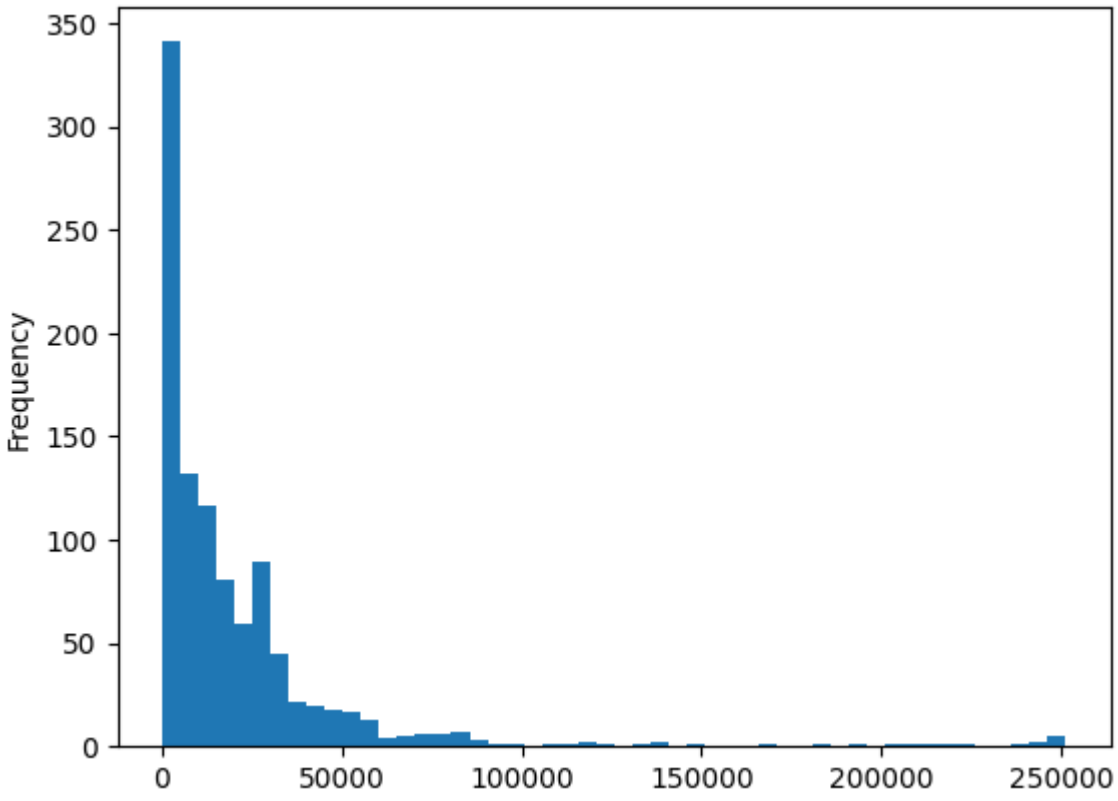
In [9]:
```
for col in dataset.columns:
    pct_missing = np.mean(dataset[col].isnull())
    if pct_missing > 0:
        print('{}: {} - {}%'.format(col, dataset[col].isna().sum(), round(pct_missi
```

```
GRADES_PK_G: 8 - 0.78%
```

Осталось лишь заполнить пропуски в столбце GRADES_PK_G. Посмотрим гистограмму распределения его значений.

In [10]:
```
dataset['GRADES_PK_G'].plot.hist(bins=50)
```

Out[10]:
```
<Axes: ylabel='Frequency'>
```



Наиболее оптимальной стратегией в данном случае является заполнение наиболее часто встречающимся значением.

In [11]:
```
imputer = SimpleImputer(strategy='most_frequent', missing_values=nan)
imputer = imputer.fit(dataset[['GRADES_PK_G']])
dataset['GRADES_PK_G'] = imputer.transform(dataset[['GRADES_PK_G']])
```

In [12]:
```
dataset.isna().sum()
```

Out[12]:
```
STATE                           0
YEAR                            0
ENROLL                          0
TOTAL_REVENUE                   0
FEDERAL_REVENUE                 0
STATE_REVENUE                   0
LOCAL_REVENUE                   0
TOTAL_EXPENDITURE               0
INSTRUCTION_EXPENDITURE         0
SUPPORT_SERVICES_EXPENDITURE    0
OTHER_EXPENDITURE               0
CAPITAL_OUTLAY_EXPENDITURE      0
GRADES_PK_G                     0
GRADES_KG_G                     0
GRADES_4_G                      0
GRADES_8_G                      0
GRADES_12_G                     0
GRADES_1_8_G                    0
GRADES_9_12_G                   0
GRADES_ALL_G                    0
dtype: int64
```

In [13]:
```
dataset.shape
```

Out[13]:
```
(1020, 20)
```

Кодирование признаков и разделение выборки.

В качестве целевого признака возьмем ENROLL. Закодируем столбец STATE с названиями штатов при помощи LabelEncoder.

In [14]:
```
le = LabelEncoder()
dataset['STATE']= le.fit_transform(dataset['STATE'])
```

In [15]:
```
X = dataset.drop(columns="ENROLL")
y = dataset["ENROLL"]
```

Обучение модели методом опорных векторов и оценка её качества.

В качестве метрик возьмём:

1. MSE - чтобы подчеркнуть большие ошибки
2. Median Absolute Error - чтобы оценить качество модели с устойчивостью к выбросам
3. R2 - чтобы точно и наглядно интерпретировать качество модели

In [16]:
```
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Создание модели SVM
```

```
svm_model = SVC(kernel='linear')

# Обучение модели
svm_model.fit(X_train, y_train)

# Прогнозирование классов для тестовых данных
y_pred = svm_model.predict(X_test)
```

In [17]:
```
# Оценка точности модели
mse_svr = mean_squared_error(y_test, y_pred)
mse_svr
```

Out[17]:
```
68949244063.4755
```

In [18]:
```
med_svr = median_absolute_error(y_test, y_pred)
med_svr
```

Out[18]:
```
4652.5
```

In [19]:
```
r2_svr = r2_score(y_test, y_pred)
r2_svr
```

Out[19]:
```
0.9601329071620861
```

Обучение модели случайного леса и оценка её качества.

Метрики аналогичные.

In [20]:
```
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Создание модели случайного леса
rf_model = RandomForestClassifier(n_estimators=100)

# Обучение модели
rf_model.fit(X_train, y_train)

# Прогнозирование классов для тестовых данных
y_pred = rf_model.predict(X_test)
```

In [21]:
```
# Оценка точности модели
mse_rf = mean_squared_error(y_test, y_pred)
mse_rf
```

Out[21]:
```
541734489.3039216
```

In [22]:
```
med_rf = median_absolute_error(y_test, y_pred)
med_rf
```

Out[22]:
```
3407.5
```

In [23]:
```
r2_rf = r2_score(y_test, y_pred)
r2_rf
```

Out[23]: 0.9996867640904272

Сравним качество 2-ух моделей.

In [24]:
```
print('--------------------------- MSE ---------------------------')
print('SVM:          ', mse_svr)
print('RandomForest: ', mse_rf)
print('--------------------------- MedAE ---------------------------')
print('SVM:          ', med_svr)
print('RandomForest: ', med_rf)
print('--------------------------- R2 ---------------------------')
print('SVM:          ', r2_svr)
print('RandomForest: ', r2_rf)
```

```
--------------------------- MSE ---------------------------
SVM:           68949244063.4755
RandomForest:  541734489.3039216
--------------------------- MedAE ---------------------------
SVM:           4652.5
RandomForest:  3407.5
--------------------------- R2 ---------------------------
SVM:           0.9601329071620861
RandomForest:  0.9996867640904272
```

Вывод.

Обе модели получились очень точными, что показывает практически единичный коэффициент детерминации. Модель Случайного леса оказалась немного более устойчивой к выбросам в данных, что показывает разница на два порядка в метрике MSE, а также в целом немного точнее модели Линейной регрессии, что также показывает метрика MedAE. Такой высокой точности удалось добиться из-за сильной корреляции в признаках выборки, а также из-за малого объёма выборки.