|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_Системы обработки информации и управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

***\_\_\_\_\_\_\_Решение задачи машинного обучения\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_\_\_ИУ5-64б\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Кан А.Д.\_\_\_\_\_\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Гапанюк Ю. Е.\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Гапанюк Ю. Е.\_**

(Подпись, дата) (И.О.Фамилия)

*2021 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_ИУ5\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_В. М. Чёрненький\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине \_\_\_\_\_\_\_\_\_\_\_Технологии машинного обучения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_\_ИУ5-64б\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кан Андрей Дмитриевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Тема курсового проекта \_\_\_\_\_\_\_\_Решение задачи машинного обучения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направленность КП: учебная

Источник тематики: кафедра

График выполнения проекта: 25% к \_10\_ нед., 50% к \_12\_ нед., 75% к 15 нед., 100% к \_17\_ нед.

***Задание\_\_\_\_Типовое исследование - решение задачи машинного обучения на основе материалов дисциплины. Исследование набора данных, обучение базового решения, подбор гиперпараметров для выбранных моделей. Формирование выводов о качестве построенных моделей\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Оформление курсового проекта:***

Расчетно-пояснительная записка на \_\_\_\_\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания « \_8\_ » \_мая\_\_\_\_\_\_\_\_ 2021 г.

**Руководитель курсового проекта**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_**Гапанюк Ю. Е.**\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Кан А.Д.\_\_**

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

## Jupyter Notebook с анализом моделей МО.

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LinearRegression, LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier

from sklearn.metrics import accuracy\_score, balanced\_accuracy\_score

from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import plot\_confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_squared\_log\_error, median\_absolute\_error, r2\_score

from sklearn.metrics import roc\_curve, roc\_auc\_score

from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export\_graphviz

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor

from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor

from gmdhpy import gmdh

%matplotlib inline

sns.set(style="ticks")

Наш датасет - admission rate. Признаки:

* GRE Scores ( out of 340 )
* TOEFL Scores ( out of 120 )
* University Rating ( out of 5 )
* Statement of Purpose Strength( out of 5 )
* Letter of Recommendation Strength ( out of 5 )
* Undergraduate GPA ( out of 10 )
* Research Experience ( either 0 or 1 )
* Chance of Admission ( ranging from 0 to 1 )

Целевым признаком выберем Chance of admission.

## Загрузка и анализ данных.

pre\_data = pd.read\_csv('Admission\_Predict.csv', sep=",")

pre\_data.shape

(400, 9)

# проверим есть ли пропущенные значения

pre\_data.isnull().sum()

Serial No. 0

GRE Score 0

TOEFL Score 0

University Rating 0

SOP 0

LOR 0

CGPA 0

Research 0

Chance of Admit 0

dtype: int64

pre\_data

|  | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| 1 | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| 2 | 3 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| 3 | 4 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| 4 | 5 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 395 | 396 | 324 | 110 | 3 | 3.5 | 3.5 | 9.04 | 1 | 0.82 |
| 396 | 397 | 325 | 107 | 3 | 3.0 | 3.5 | 9.11 | 1 | 0.84 |
| 397 | 398 | 330 | 116 | 4 | 5.0 | 4.5 | 9.45 | 1 | 0.91 |
| 398 | 399 | 312 | 103 | 3 | 3.5 | 4.0 | 8.78 | 0 | 0.67 |
| 399 | 400 | 333 | 117 | 4 | 5.0 | 4.0 | 9.66 | 1 | 0.95 |

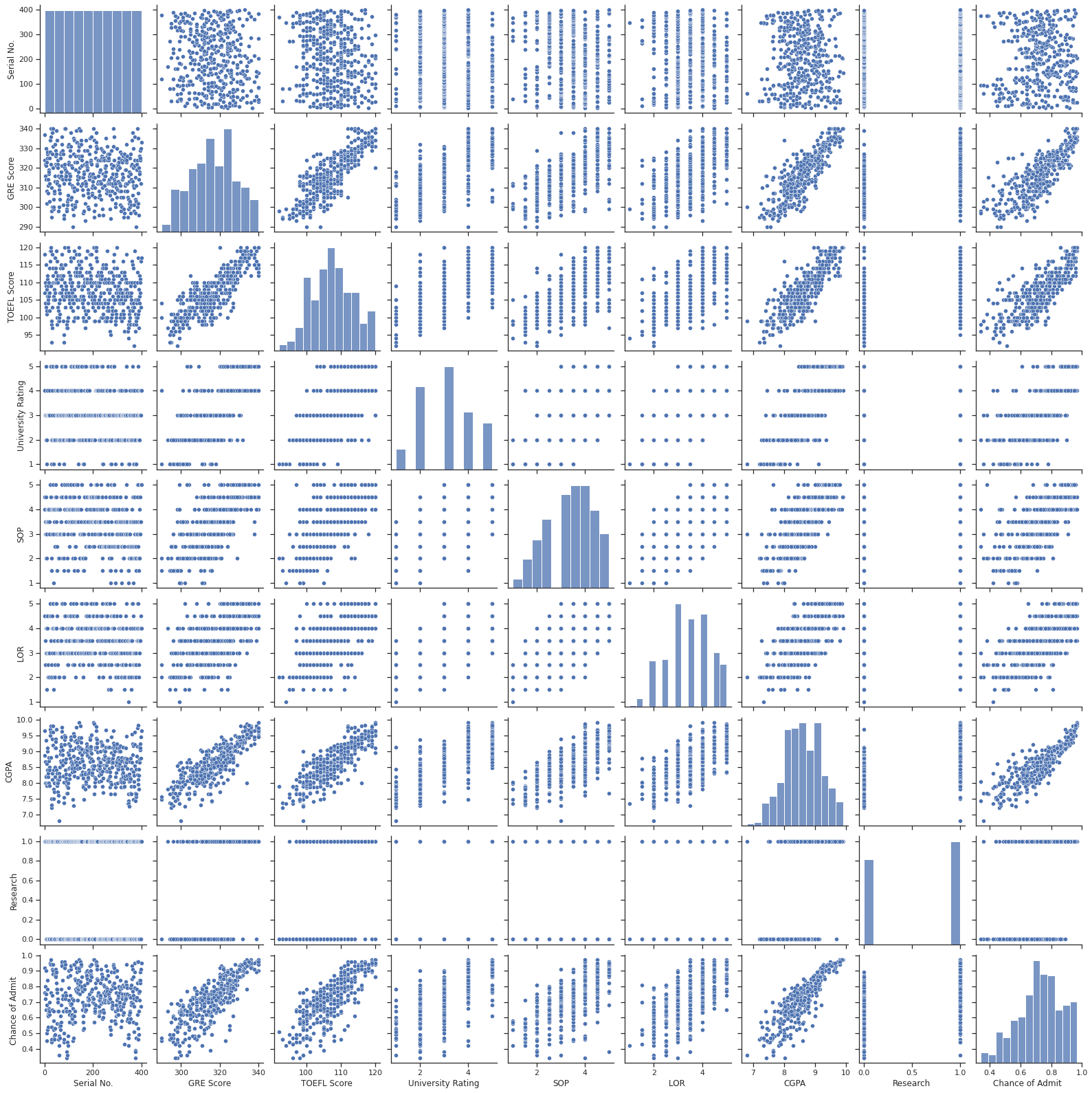
400 rows × 9 columns

## Построение графиков для понимания структуры данных

# Парные диаграммы

sns.pairplot(pre\_data)

<seaborn.axisgrid.PairGrid at 0x7f7c1a287c70>



pre\_data.columns

Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',

'LOR ', 'CGPA', 'Research', 'Chance of Admit '],

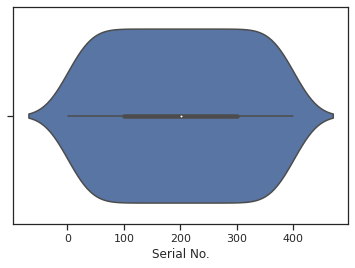
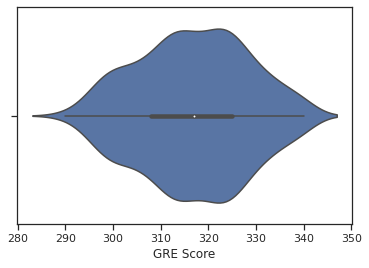
dtype='object')

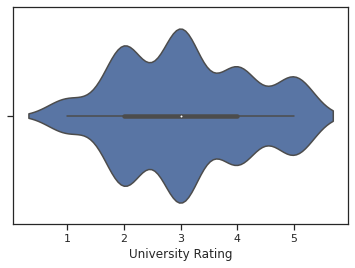
# Скрипичные диаграммы для числовых колонок

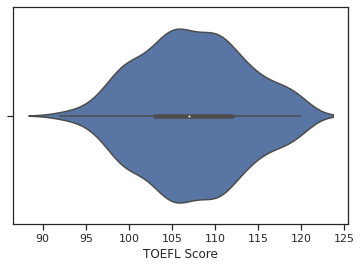
for col in pre\_data.columns:

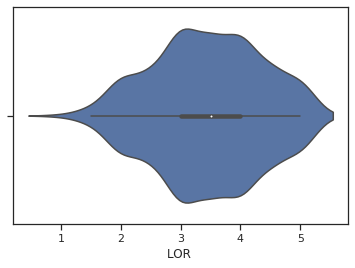
sns.violinplot(x=pre\_data[col])

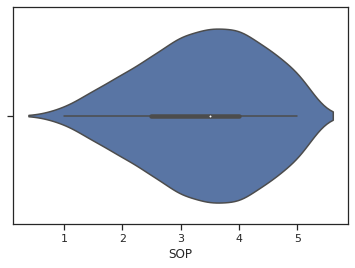
plt.show()

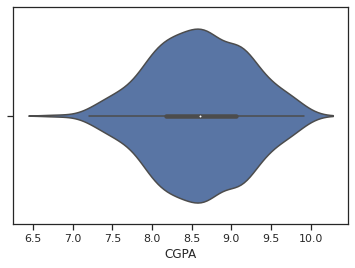
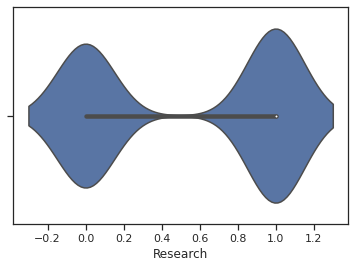


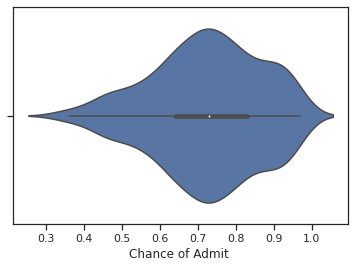












## Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

pre\_data.dtypes

Serial No. int64

GRE Score int64

TOEFL Score int64

University Rating int64

SOP float64

LOR float64

CGPA float64

Research int64

Chance of Admit float64

dtype: object

Категориальные признаки отсутствуют, их кодирования не требуется.

Вспомогательные признаки для улучшения качества моделей в данном примере мы строить не будем.

Выполним масштабирование данных.

# Числовые колонки для масштабирования

scale\_cols = ['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',

'LOR ', 'CGPA', 'Research', 'Chance of Admit ']

sc1 = MinMaxScaler()

sc1\_data = sc1.fit\_transform(pre\_data[scale\_cols])

data\_all = pre\_data.copy()

# Добавим масштабированные данные в набор данных

for i in range(len(scale\_cols)):

col = scale\_cols[i]

new\_col\_name = col + '\_scaled'

data\_all[new\_col\_name] = sc1\_data[:,i]

data\_all

|  | Serial No. | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit | Serial No.\_scaled | GRE Score\_scaled | TOEFL Score\_scaled | University Rating\_scaled | SOP\_scaled | LOR \_scaled | CGPA\_scaled | Research\_scaled | Chance of Admit \_scaled |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 | 0.000000 | 0.94 | 0.928571 | 0.75 | 0.875 | 0.875 | 0.913462 | 1.0 | 0.920635 |
| 1 | 2 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 | 0.002506 | 0.68 | 0.535714 | 0.75 | 0.750 | 0.875 | 0.663462 | 1.0 | 0.666667 |
| 2 | 3 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 | 0.005013 | 0.52 | 0.428571 | 0.50 | 0.500 | 0.625 | 0.384615 | 1.0 | 0.603175 |
| 3 | 4 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 | 0.007519 | 0.64 | 0.642857 | 0.50 | 0.625 | 0.375 | 0.599359 | 1.0 | 0.730159 |
| 4 | 5 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 | 0.010025 | 0.48 | 0.392857 | 0.25 | 0.250 | 0.500 | 0.451923 | 0.0 | 0.492063 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 395 | 396 | 324 | 110 | 3 | 3.5 | 3.5 | 9.04 | 1 | 0.82 | 0.989975 | 0.68 | 0.642857 | 0.50 | 0.625 | 0.625 | 0.717949 | 1.0 | 0.761905 |
| 396 | 397 | 325 | 107 | 3 | 3.0 | 3.5 | 9.11 | 1 | 0.84 | 0.992481 | 0.70 | 0.535714 | 0.50 | 0.500 | 0.625 | 0.740385 | 1.0 | 0.793651 |
| 397 | 398 | 330 | 116 | 4 | 5.0 | 4.5 | 9.45 | 1 | 0.91 | 0.994987 | 0.80 | 0.857143 | 0.75 | 1.000 | 0.875 | 0.849359 | 1.0 | 0.904762 |
| 398 | 399 | 312 | 103 | 3 | 3.5 | 4.0 | 8.78 | 0 | 0.67 | 0.997494 | 0.44 | 0.392857 | 0.50 | 0.625 | 0.750 | 0.634615 | 0.0 | 0.523810 |
| 399 | 400 | 333 | 117 | 4 | 5.0 | 4.0 | 9.66 | 1 | 0.95 | 1.000000 | 0.86 | 0.892857 | 0.75 | 1.000 | 0.750 | 0.916667 | 1.0 | 0.968254 |

400 rows × 18 columns

# Проверим, что масштабирование не повлияло на распределение данных

for col in scale\_cols:

col\_scaled = col + '\_scaled'

fig, ax = plt.subplots(1, 2, figsize=(8,3))

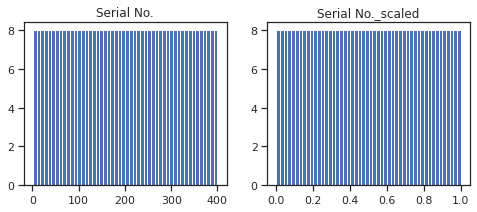
ax[0].hist(data\_all[col], 50)

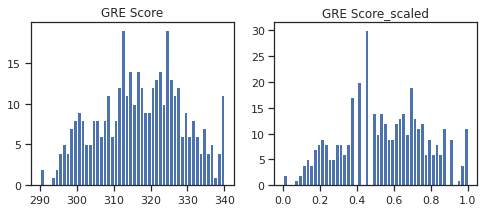
ax[1].hist(data\_all[col\_scaled], 50)

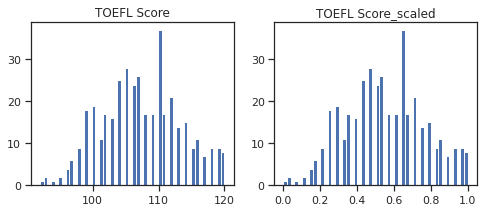
ax[0].title.set\_text(col)

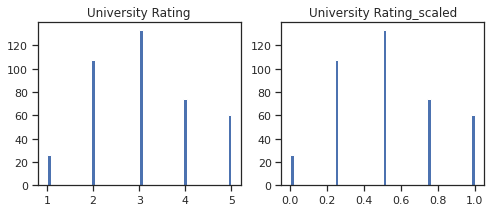
ax[1].title.set\_text(col\_scaled)

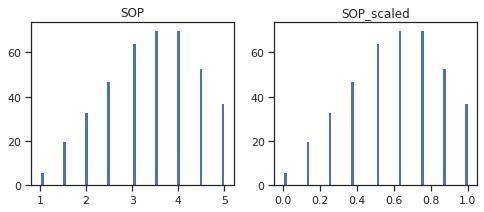
plt.show()

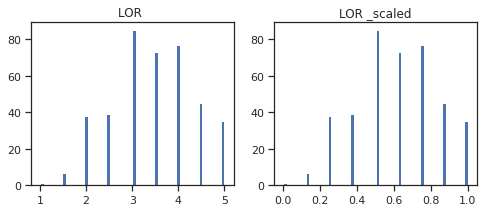


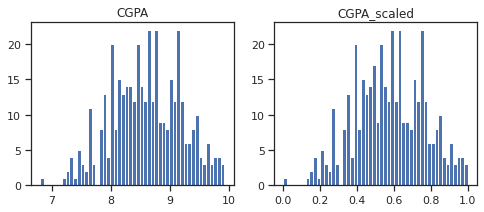


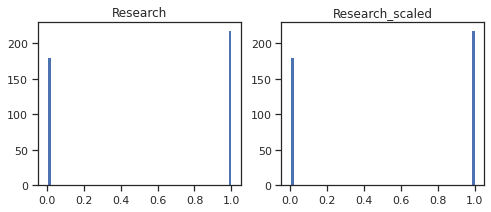


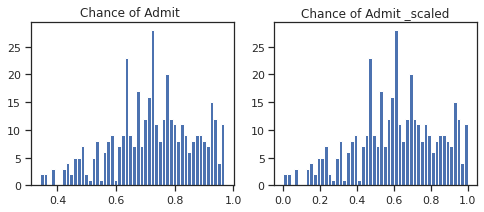












## Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

corr\_cols\_1 = scale\_cols

corr\_cols\_1

['Serial No.',

'GRE Score',

'TOEFL Score',

'University Rating',

'SOP',

'LOR ',

'CGPA',

'Research',

'Chance of Admit ']

scale\_cols\_postfix = [x+'\_scaled' for x in scale\_cols]

corr\_cols\_2 = scale\_cols\_postfix

corr\_cols\_2

['Serial No.\_scaled',

'GRE Score\_scaled',

'TOEFL Score\_scaled',

'University Rating\_scaled',

'SOP\_scaled',

'LOR \_scaled',

'CGPA\_scaled',

'Research\_scaled',

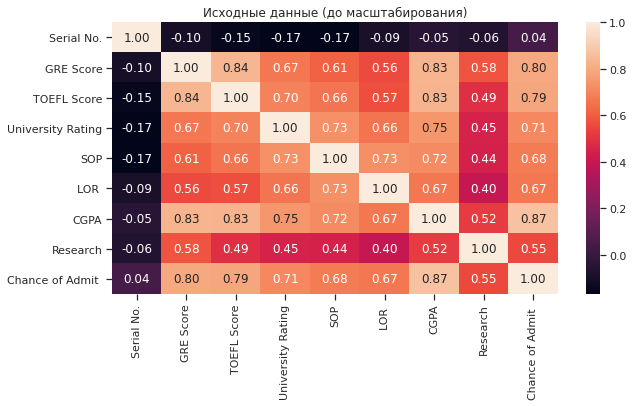
'Chance of Admit \_scaled']

fig, ax = plt.subplots(figsize=(10,5))

sns.heatmap(data\_all[corr\_cols\_1].corr(), annot=True, fmt='.2f')

ax.set\_title('Исходные данные (до масштабирования)')

plt.show()

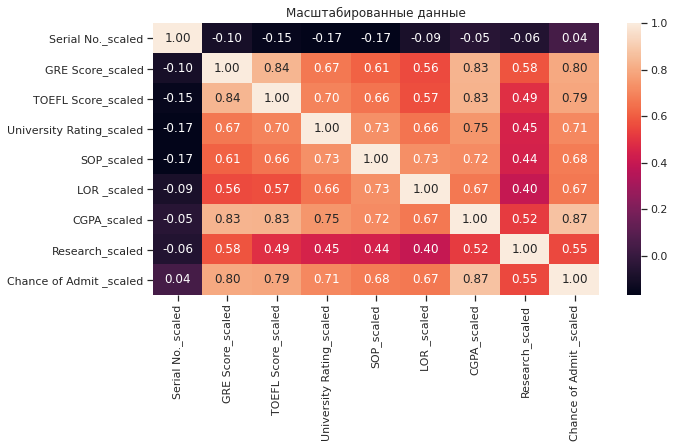


fig, ax = plt.subplots(figsize=(10,5))

sns.heatmap(data\_all[corr\_cols\_2].corr(), annot=True, fmt='.2f')

ax.set\_title('Масштабированные данные')

plt.show()



На основе корреляционной матрицы можно сделать следующие выводы:

* Видим, что Serial No. является лишним признаком. Это просто индекс, который никак не влияет на модель.
* Остальные признаки хорошо коррелируют с целевым признаком

# Удаляем ненужные для построения модели признаки

data = pre\_data.drop(["Serial No."], axis=1)

data

|  | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research | Chance of Admit |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 | 0.92 |
| 1 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 | 0.76 |
| 2 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 | 0.72 |
| 3 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 | 0.80 |
| 4 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 | 0.65 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 395 | 324 | 110 | 3 | 3.5 | 3.5 | 9.04 | 1 | 0.82 |
| 396 | 325 | 107 | 3 | 3.0 | 3.5 | 9.11 | 1 | 0.84 |
| 397 | 330 | 116 | 4 | 5.0 | 4.5 | 9.45 | 1 | 0.91 |
| 398 | 312 | 103 | 3 | 3.5 | 4.0 | 8.78 | 0 | 0.67 |
| 399 | 333 | 117 | 4 | 5.0 | 4.0 | 9.66 | 1 | 0.95 |

400 rows × 8 columns

## Формирование обучающей и тестовой выборок на основе исходного набора данных.

data\_X = data.drop(["Chance of Admit "], axis=1)

data\_X

|  | GRE Score | TOEFL Score | University Rating | SOP | LOR | CGPA | Research |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 337 | 118 | 4 | 4.5 | 4.5 | 9.65 | 1 |
| 1 | 324 | 107 | 4 | 4.0 | 4.5 | 8.87 | 1 |
| 2 | 316 | 104 | 3 | 3.0 | 3.5 | 8.00 | 1 |
| 3 | 322 | 110 | 3 | 3.5 | 2.5 | 8.67 | 1 |
| 4 | 314 | 103 | 2 | 2.0 | 3.0 | 8.21 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 395 | 324 | 110 | 3 | 3.5 | 3.5 | 9.04 | 1 |
| 396 | 325 | 107 | 3 | 3.0 | 3.5 | 9.11 | 1 |
| 397 | 330 | 116 | 4 | 5.0 | 4.5 | 9.45 | 1 |
| 398 | 312 | 103 | 3 | 3.5 | 4.0 | 8.78 | 0 |
| 399 | 333 | 117 | 4 | 5.0 | 4.0 | 9.66 | 1 |

400 rows × 7 columns

data\_Y = data[["Chance of Admit "]]

data\_Y

|  | Chance of Admit |
| --- | --- |
| 0 | 0.92 |
| 1 | 0.76 |
| 2 | 0.72 |
| 3 | 0.80 |
| 4 | 0.65 |
| ... | ... |
| 395 | 0.82 |
| 396 | 0.84 |
| 397 | 0.91 |
| 398 | 0.67 |
| 399 | 0.95 |

400 rows × 1 columns

sc = MinMaxScaler()

data\_X\_sc = sc.fit\_transform(data\_X)

data\_Y\_sc = sc.fit\_transform(data\_Y)

# Разделим данные на тестовую и тренировочную

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data\_X\_sc, data\_Y\_sc, test\_size=0.33, random\_state=1)

## Выбор метрик для последующей оценки качества моделей.

### В качестве метрик для решения задачи регрессии будем использовать:

* Mean absolute error - средняя абсолютная ошибка.
* Mean squared error - средняя квадратичная ошибка.
* Метрика R2 или коэффициент детерминации

### Сохранение и визуализация метрик

class MetricLogger:

def \_\_init\_\_(self):

self.df = pd.DataFrame(

{'metric': pd.Series([], dtype='str'),

'alg': pd.Series([], dtype='str'),

'value': pd.Series([], dtype='float')})

def add(self, metric, alg, value):

"""

Добавление значения

"""

# Удаление значения если оно уже было ранее добавлено

self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)

# Добавление нового значения

temp = [{'metric':metric, 'alg':alg, 'value':value}]

self.df = self.df.append(temp, ignore\_index=True)

def get\_data\_for\_metric(self, metric, ascending=True):

"""

Формирование данных с фильтром по метрике

"""

temp\_data = self.df[self.df['metric']==metric]

temp\_data\_2 = temp\_data.sort\_values(by='value', ascending=ascending)

return temp\_data\_2['alg'].values, temp\_data\_2['value'].values

def plot(self, str\_header, metric, ascending=True, figsize=(5, 5)):

"""

Вывод графика

"""

array\_labels, array\_metric = self.get\_data\_for\_metric(metric, ascending)

fig, ax1 = plt.subplots(figsize=figsize)

pos = np.arange(len(array\_metric))

rects = ax1.barh(pos, array\_metric,

align='center',

height=0.5,

tick\_label=array\_labels)

ax1.set\_title(str\_header)

for a,b in zip(pos, array\_metric):

plt.text(0.5, a-0.05, str(round(b,3)), color='white')

plt.show()

## Выбор наиболее подходящих моделей для решения задачи регрессии.

Для задачи регрессии будем использовать следующие модели:

* Линейная регрессия
* Метод ближайших соседей
* Решающее дерево
* Случайный лес
* Градиентный бустинг

## Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

# Модели

regr\_models = {'LR': LinearRegression(),

'KNN\_5':KNeighborsRegressor(n\_neighbors=5),

'Tree':DecisionTreeRegressor(),

'RF':RandomForestRegressor(),

'GB':GradientBoostingRegressor()}

# Сохранение метрик

regrMetricLogger = MetricLogger()

def regr\_train\_model(model\_name, model, regrMetricLogger):

model.fit(X\_train, Y\_train)

Y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(Y\_test, Y\_pred)

mse = mean\_squared\_error(Y\_test, Y\_pred)

r2 = r2\_score(Y\_test, Y\_pred)

regrMetricLogger.add('MAE', model\_name, mae)

regrMetricLogger.add('MSE', model\_name, mse)

regrMetricLogger.add('R2', model\_name, r2)

print('{} \t MAE={}, MSE={}, R2={}'.format(

model\_name, round(mae, 3), round(mse, 3), round(r2, 3)))

for model\_name, model in regr\_models.items():

regr\_train\_model(model\_name, model, regrMetricLogger)

LR MAE=0.072, MSE=0.011, R2=0.804

KNN\_5 MAE=0.091, MSE=0.018, R2=0.684

Tree MAE=0.107, MSE=0.023, R2=0.598

RF MAE=0.078, MSE=0.013, R2=0.783

GB MAE=0.08, MSE=0.014, R2=0.764

<ipython-input-92-53aa041513ca>:2: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n\_samples,), for example using ravel().

model.fit(X\_train, Y\_train)

/home/ripper/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py:72: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n\_samples, ), for example using ravel().

return f(\*\*kwargs)

## Подбор гиперпараметров для выбранных моделей.

### Метод ближайших соседей

n\_range = np.array(range(1,200,2))

tuned\_parameters = [{'n\_neighbors': n\_range}]

tuned\_parameters

[{'n\_neighbors': array([ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25,

27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51,

53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77,

79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103,

105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129,

131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155,

157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181,

183, 185, 187, 189, 191, 193, 195, 197, 199])}]

%%time

regr\_gs = GridSearchCV(KNeighborsRegressor(), tuned\_parameters, cv=5, scoring='neg\_mean\_squared\_error')

regr\_gs.fit(X\_train, Y\_train)

CPU times: user 1.95 s, sys: 26.8 ms, total: 1.98 s

Wall time: 840 ms

GridSearchCV(cv=5, estimator=KNeighborsRegressor(),

param\_grid=[{'n\_neighbors': array([ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25,

27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51,

53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77,

79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103,

105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129,

131, 133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155,

157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181,

183, 185, 187, 189, 191, 193, 195, 197, 199])}],

scoring='neg\_mean\_squared\_error')

# Лучшая модель

-regr\_gs.best\_score\_

0.012448597212291529

# Лучшее значение параметров

regr\_gs.best\_params\_

{'n\_neighbors': 15}

regr\_gs\_best\_params\_txt = str(regr\_gs.best\_params\_['n\_neighbors'])

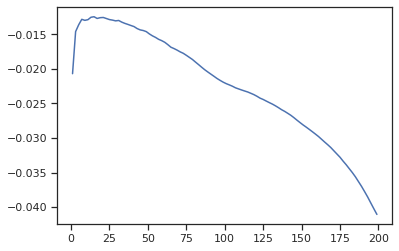
regr\_gs\_best\_params\_txt

'15'

# Изменение качества на тестовой выборке в зависимости от К-соседей

plt.plot(n\_range, regr\_gs.cv\_results\_['mean\_test\_score'])

[<matplotlib.lines.Line2D at 0x7f7c065520a0>]



### Дерево решений

params = {

'max\_depth': [3, 4, 5, 6],

'min\_samples\_leaf': [0.04, 0.06, 0.08],

'max\_features': [0.2, 0.4, 0.6, 0.8]

}

%%time

regr\_gs1 = GridSearchCV(estimator=DecisionTreeRegressor(random\_state=1),

param\_grid=params, scoring='neg\_mean\_squared\_error', cv=3, n\_jobs=-1)

regr\_gs1.fit(X\_train, Y\_train)

CPU times: user 173 ms, sys: 80.1 ms, total: 253 ms

Wall time: 1.15 s

GridSearchCV(cv=3, estimator=DecisionTreeRegressor(random\_state=1), n\_jobs=-1,

param\_grid={'max\_depth': [3, 4, 5, 6],

'max\_features': [0.2, 0.4, 0.6, 0.8],

'min\_samples\_leaf': [0.04, 0.06, 0.08]},

scoring='neg\_mean\_squared\_error')

# Лучшая модель

-regr\_gs1.best\_score\_

0.014119433623414899

regr\_gs\_best\_params\_txt1 = str(regr\_gs1.best\_params\_)

regr\_gs\_best\_params\_txt1

"{'max\_depth': 3, 'max\_features': 0.8, 'min\_samples\_leaf': 0.06}"

# Лучшее значение параметров

regr\_gs1.best\_params\_

{'max\_depth': 3, 'max\_features': 0.8, 'min\_samples\_leaf': 0.06}

## Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

### Метод ближайших соседей

regr\_models\_grid = {'KNN\_5':KNeighborsRegressor(n\_neighbors=5),

str('KNN\_'+regr\_gs\_best\_params\_txt):regr\_gs.best\_estimator\_}

for model\_name, model in regr\_models\_grid.items():

regr\_train\_model(model\_name, model, regrMetricLogger)

KNN\_5 MAE=0.091, MSE=0.018, R2=0.684

KNN\_15 MAE=0.09, MSE=0.018, R2=0.685

### Дерево решений

regr\_models\_grid = {'Tree':DecisionTreeRegressor(random\_state=1),

str('Tree\_'+regr\_gs\_best\_params\_txt1):regr\_gs1.best\_estimator\_}

for model\_name, model in regr\_models\_grid.items():

regr\_train\_model(model\_name, model, regrMetricLogger)

Tree MAE=0.102, MSE=0.021, R2=0.632

Tree\_{'max\_depth': 3, 'max\_features': 0.8, 'min\_samples\_leaf': 0.06} MAE=0.094, MSE=0.017, R2=0.704

## Формирование выводов о качестве построенных моделей на основе выбранных метрик.

# Метрики качества модели

regr\_metrics = regrMetricLogger.df['metric'].unique()

regr\_metrics

array(['MAE', 'MSE', 'R2'], dtype=object)

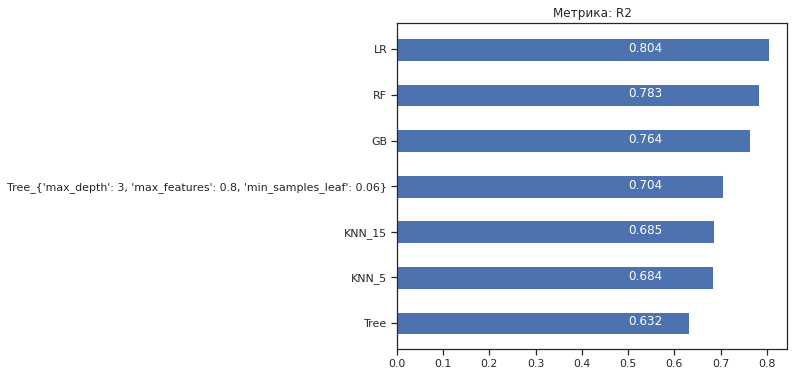
regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(7, 6))



regrMetricLogger.plot('Метрика: ' + 'MSE', 'MSE', ascending=False, figsize=(7, 6))



regrMetricLogger.plot('Метрика: ' + 'R2', 'R2', ascending=True, figsize=(7, 6))



## Вывод

Лучшей моделью оказалась линейная регрессия. Это связано с тем, что в датасете огромное количество линейных зависимостей. За ней следует случайное дерево. Несмотря на то, что подбор гиперпараметров помог улучшить качество 2 моделей, они не смогли сравниться со случайным лесом и линейной регрессией.

## Исходный код веб-приложения для анализа данных.

|  |
| --- |
| 1. from sklearn.datasets import \* |
|  |

|  |
| --- |
| import streamlit as st |
|  |

|  |
| --- |
| import seaborn as sns |
|  |

|  |
| --- |
| import pandas as pd |
|  |

|  |
| --- |
| import numpy as np |
|  |

|  |
| --- |
| from sklearn.preprocessing import MinMaxScaler |
|  |

|  |
| --- |
| from sklearn.model\_selection import cross\_val\_score |
|  |

|  |
| --- |
| from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier |
|  |

|  |
| --- |
| import matplotlib.pyplot as plt |
|  |

|  |
| --- |
| from sklearn.model\_selection import train\_test\_split |
|  |

|  |
| --- |
| from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export\_graphviz |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| @st.cache |
|  |

|  |
| --- |
| def load\_data(): |
|  |

|  |
| --- |
| ''' |
|  |

|  |
| --- |
| Загрузка данных |
|  |

|  |
| --- |
| ''' |
|  |

|  |
| --- |
| pre\_data = pd.read\_csv('Admission\_Predict.csv', sep=",") |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| data = pre\_data.drop(["Serial No."], axis=1) |
|  |

|  |
| --- |
| data\_X = data.drop(["Chance of Admit "], axis=1) |
|  |

|  |
| --- |
| data\_Y = data[["Chance of Admit "]] |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| sc = MinMaxScaler() |
|  |

|  |
| --- |
| data\_X\_sc = sc.fit\_transform(data\_X) |
|  |

|  |
| --- |
| data\_Y\_sc = sc.fit\_transform(data\_Y) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return data\_X\_sc, data\_Y\_sc, data.shape[0], data |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| st.header('Обучение модели ближайших соседей') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| data\_load\_state = st.text('Загрузка данных...') |
|  |

|  |
| --- |
| data\_X, data\_Y, data\_len, data = load\_data() |
|  |

|  |
| --- |
| data\_load\_state.text('Данные загружены!') |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| st.write(data.head()) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| cv\_slider = st.slider('Количество фолдов:', min\_value=3, max\_value=10, value=5, step=1) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| #Вычислим количество возможных ближайших соседей |
|  |

|  |
| --- |
| rows\_in\_one\_fold = int(data\_len / cv\_slider) |
|  |

|  |
| --- |
| allowed\_knn = int(rows\_in\_one\_fold \* (cv\_slider-1)) |
|  |

|  |
| --- |
| st.write('Количество строк в наборе данных - {}'.format(data\_len)) |
|  |

|  |
| --- |
| st.subheader('Метод ближайших соседей') |
|  |

|  |
| --- |
| st.write('Максимальное допустимое количество ближайших соседей с учетом выбранного количества фолдов - {}'.format(allowed\_knn)) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| cv\_knn = st.slider('Количество ближайших соседей:', min\_value=1, max\_value=allowed\_knn, value=5, step=1) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| scores = cross\_val\_score(KNeighborsRegressor(n\_neighbors=cv\_knn), |
|  |

|  |
| --- |
| data\_X, data\_Y, scoring='neg\_mean\_squared\_error', cv=cv\_slider) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| st.subheader('Оценка качества модели метода ближайших соседей') |
|  |

|  |
| --- |
| st.write('Значения neg\_mean\_squared\_error для отдельных фолдов') |
|  |

|  |
| --- |
| st.bar\_chart(scores) |
|  |

|  |
| --- |
| st.write('Усредненное значение neg\_mean\_squared\_error по всем фолдам - {}'.format(-np.mean(scores))) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| st.subheader('Деревья решений') |
|  |

|  |
| --- |
| cv\_max\_depth = st.slider('Максимальная гоубина:', min\_value=3, max\_value=6, value=3, step=1) |
|  |

|  |
| --- |
| cv\_min\_samples\_leaf = st.slider('Минимальное количество листьев:', min\_value=0.04, max\_value=0.08, value=0.04, step=0.02) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| scores1 = cross\_val\_score(DecisionTreeRegressor(random\_state=1, max\_depth=cv\_max\_depth, min\_samples\_leaf=cv\_min\_samples\_leaf), |
|  |

|  |
| --- |
| data\_X, data\_Y, scoring='neg\_mean\_squared\_error', cv=cv\_slider) |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| st.subheader('Оценка качества модели деревья решений') |
|  |

|  |
| --- |
| st.write('Значения neg\_mean\_squared\_error для отдельных фолдов') |
|  |

|  |
| --- |
| st.bar\_chart(scores1) |
|  |

st.write('Усредненное значение neg\_mean\_squared\_error по всем фолдам - {}'.format(-np.mean(scores)))

## Демонстрация работы веб-приложения.

