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)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

_____

_____

_____

_____

_____

_____

Студент: РТ5-61 _____ **Коржов Сергей Юрьевич**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **Гапанюк Юрий Евгеньевич**
(Подпись, дата) (И.О.Фамилия)

2020 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)

ЗАДАНИЕ
на выполнение курсового проекта

по дисциплине Технологии машинного обучения

Студент группы РТ5-61Б

Коржов Сергей Юрьевич

Тема курсового проекта _____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____

Источник тематики (кафедра, предприятие, НИР) _____

Задание_____
_____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на _____ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ Гапанюк Ю.Е
(Подпись, дата) (И.О.Фамилия)

Студент _____ Коржов С.Ю.
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

1. Введение

Данный курсовой проект направлен на решение комплексной задачи машинного обучения. Предстоит выполнить типовую задачу машинного обучения – провести анализ данных, провести ряд операций над датасетом, подобрать модели, а также наиболее подходящие гиперпараметры выбранных моделей.

Современные условия, в которых функционируют информационные системы, предполагают использование неструктурированных данных и эффективных средств для работы с ними. Они собирают и обрабатывают огромные объемы быстро поступающей цифровой информации и анализируют её, находя определенные закономерности, что позволяет разработать системы для классификации и прогнозирования.

Поэтому машинное обучение очень актуально в современном мире, и оно широко используется. Программист должен уметь правильно применять технологии машинного обучения для достижения наилучших результатов, чему мы и научимся в этом курсовом проекте.

2. Выполнение курсового проекта

2.1. Поиск и выбор набора данных для построения моделей машинного обучения

Выбранный набор данных содержит информацию об абитуриентах, поступающих в магистратуру.

Файл содержит следующие колонки:

1. GRE Scores (340 значений) – результаты тестирования « Graduate Record Examinations»;
2. TOEFL Scores (120 значений) – результаты тестирования « Test of English as a Foreign Language»;
3. University Rating (5 значений) – рейтинг колледжей и университетов;
4. Statement of Purpose and Letter of Recommendation Strength (5 значений) – заявление о целях и рекомендательное письмо;
5. Undergraduate GPA (10 значения) – средний балл аттестата или диплома;
6. Research Experience (0 или 1) – исследовательский опыт;
7. Chance of Admit (от 0 до 1) – шанс поступления в магистратуру.

Для данного набора данных будем решать задачу регрессии. В качестве целевого признака возьмем колонку «Chance of Admit» (шанс поступления в магистратуру).

```
[329] import numpy as np
      import pandas as pd
      import seaborn as sns

      import matplotlib.pyplot as plt
      from sklearn.preprocessing import MinMaxScaler
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import roc_auc_score, precision_score, recall_score, accuracy_score, plot_confusion_matrix, roc_curve
      from sklearn.linear_model import LinearRegression
      from sklearn.neighbors import KNeighborsRegressor
      from sklearn.svm import SVR
      from sklearn.tree import DecisionTreeRegressor
      from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
      from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


      %matplotlib inline

      # Установим тип графиков
      sns.set(style="ticks")

      # Для лучшего качество графиков
      from IPython.display import set_matplotlib_formats
      set_matplotlib_formats("retina")

      # Установим ширину экрана для отчета
      pd.set_option("display.width", 80)
```

2.2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных

Загрузим данные с помощью библиотеки pandas и выведем первые 5 строк:

```
[331] # Загрузим набор данных и выведем её первые пять записей
      data = pd.read_csv('Admission_Predict.csv')
      data.head()
```

	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

Определим размер датасета, наименования колонок и типы данных, которыми заполнены колонки.

```
[ ] # Вычислим размер датасета
    data.shape
```

```
(400, 9)
```

```
[ ] # Увидим, из каких колонок состоит датасет
    data.columns
```

```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit '],
      dtype='object')
```

```
[ ] # Поймем какими типами данных заполнены колонки
    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
```

```
[ ] # Проверим наличие пустых значений
    data.isnull().sum()
```

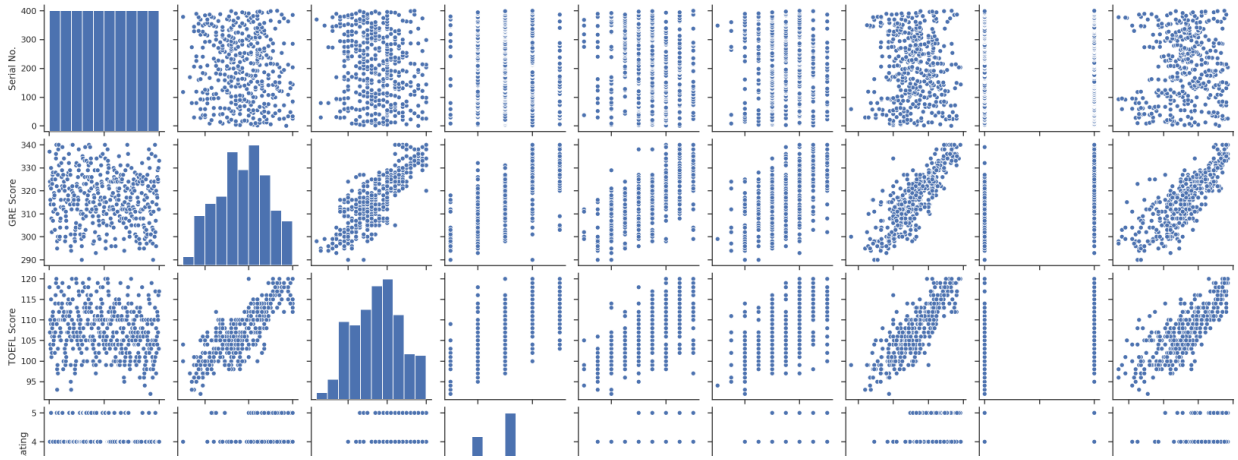
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

Пустые значения отсутствуют.

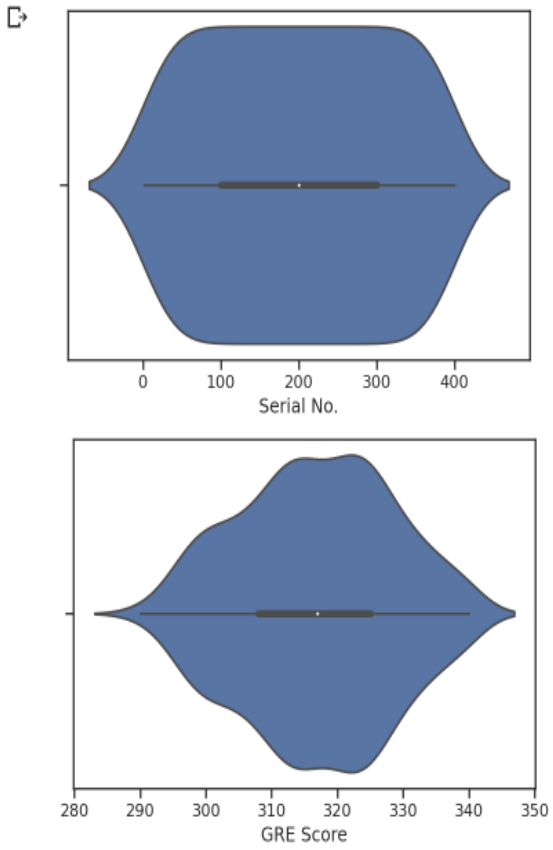
Визуализация данных.

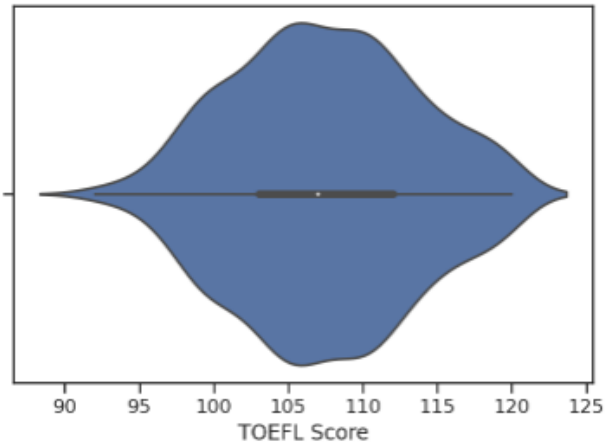
```
[ ]  # Парные диаграммы
     sns.pairplot(data)
```

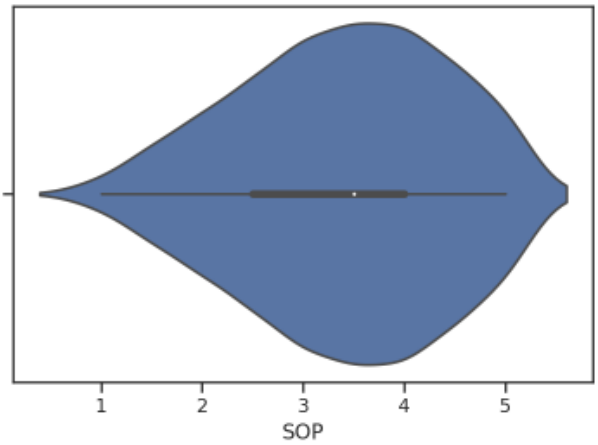
```
<seaborn.axisgrid.PairGrid at 0x7f391cfafc50>
```

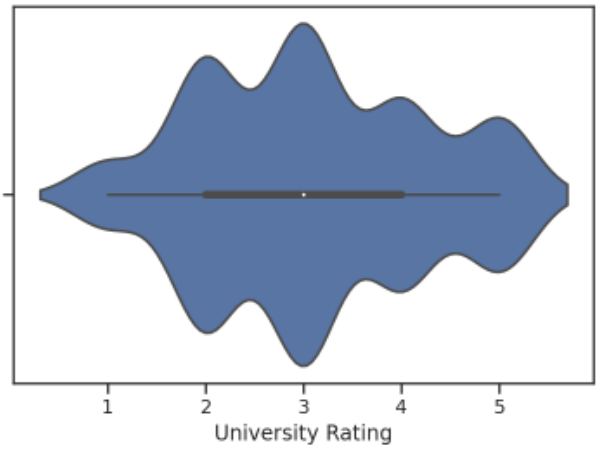


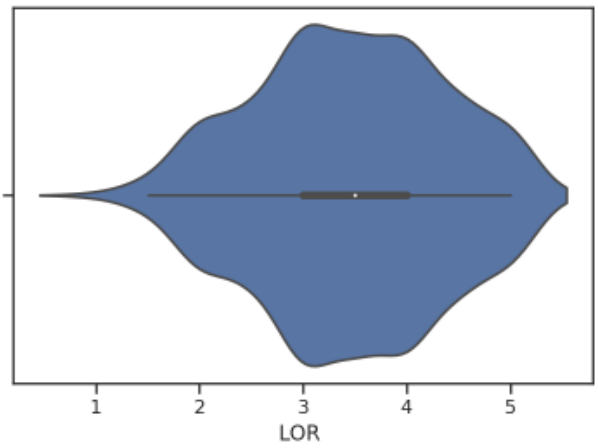
```
[ ]  # Скрипичные диаграммы для числовых колонок
     for col in ['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
            'LOR ', 'CGPA', 'Research', 'Chance of Admit ']:
         sns.violinplot(x=data[col])
         plt.show()
```

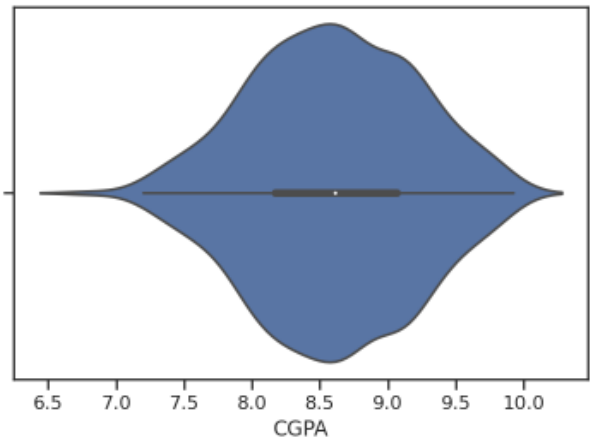


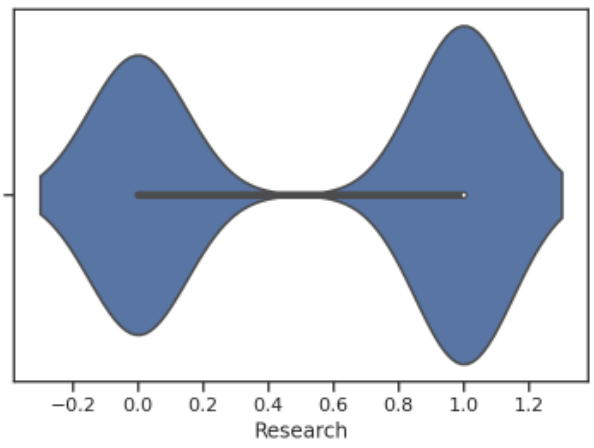

90 95 100 105 110 115 120 125
TOEFL Score

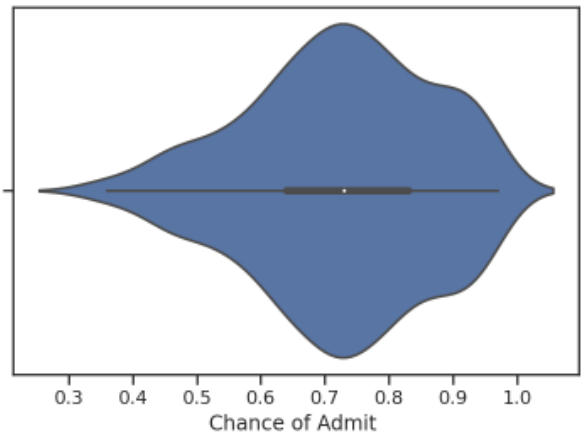

1 2 3 4 5
SOP


1 2 3 4 5
University Rating


1 2 3 4 5
LOR


6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0
CGPA


−0.2 0.0 0.2 0.4 0.6 0.8 1.0 1.2
Research


0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Chance of Admit

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

Для построения моделей будем использовать все признаки. Категориальные признаки отсутствуют, их кодирования не требуется. Вспомогательные признаки для улучшения качества моделей строить не будем.

Масштабирование данных.

```
# Числовые колонки для масштабирования
scale_cols = ['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research']
```

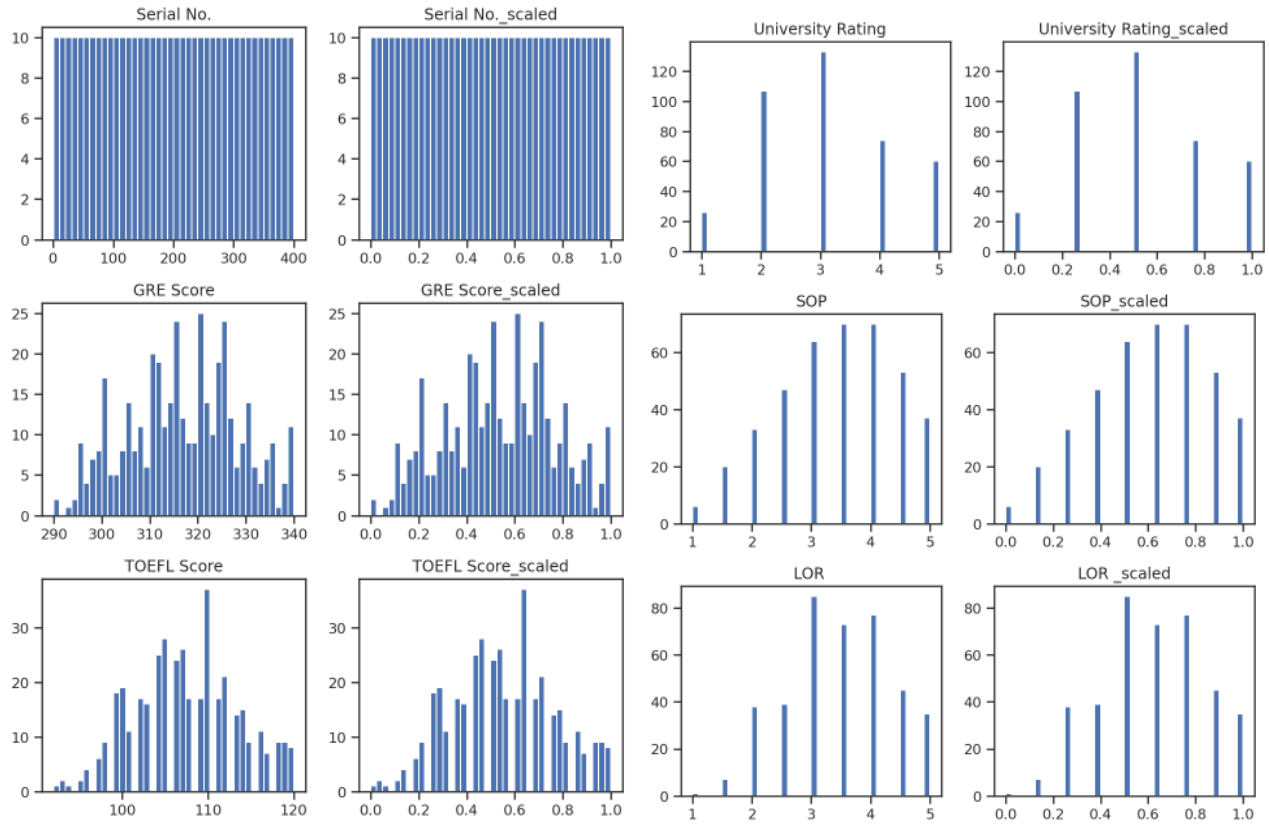
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

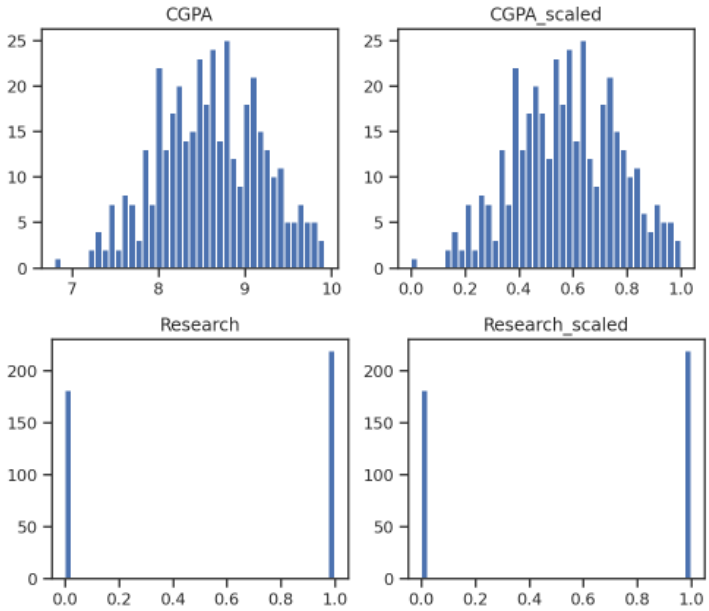
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

```
data.head()
```

	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
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
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
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
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
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

```
# 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[col], 40)
    ax[1].hist(data[col_scaled], 40)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```





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

```
[344] # Воспользуемся наличием тестовых выборок,
      # включив их в корреляционную матрицу
      corr_cols_1 = scale_cols + ['Chance of Admit ']
      corr_cols_1
```

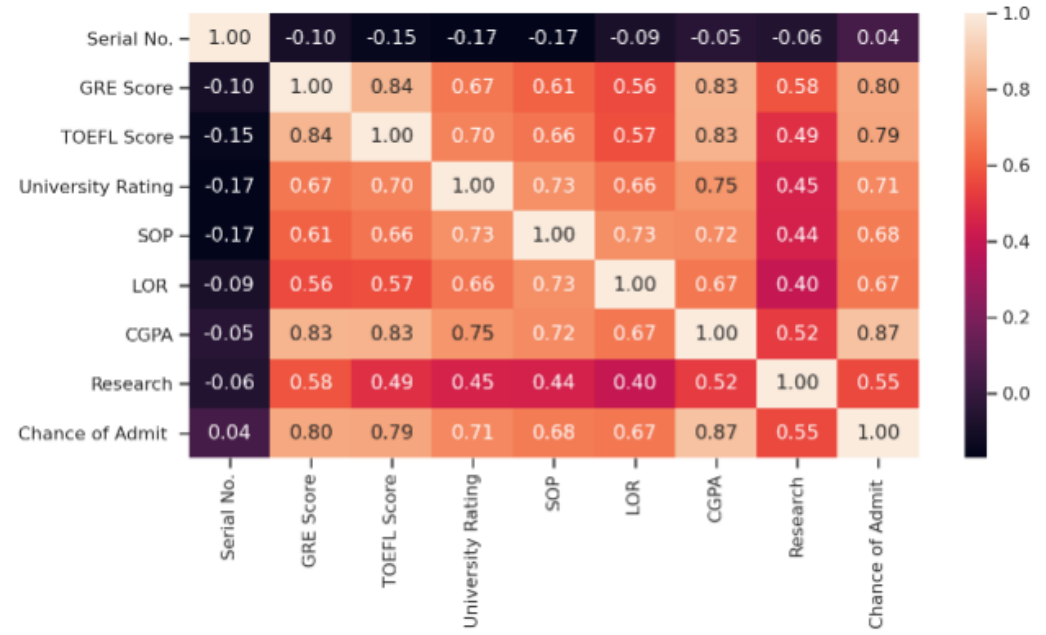
```
['Serial No.',
 'GRE Score',
 'TOEFL Score',
 'University Rating',
 'SOP',
 'LOR ',
 'CGPA',
 'Research',
 'Chance of Admit ']
```

```
[345] scale_cols_postfix = [x+'_scaled' for x in scale_cols]
      corr_cols_2 = scale_cols_postfix + ['Chance of Admit ']
      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 ']
```

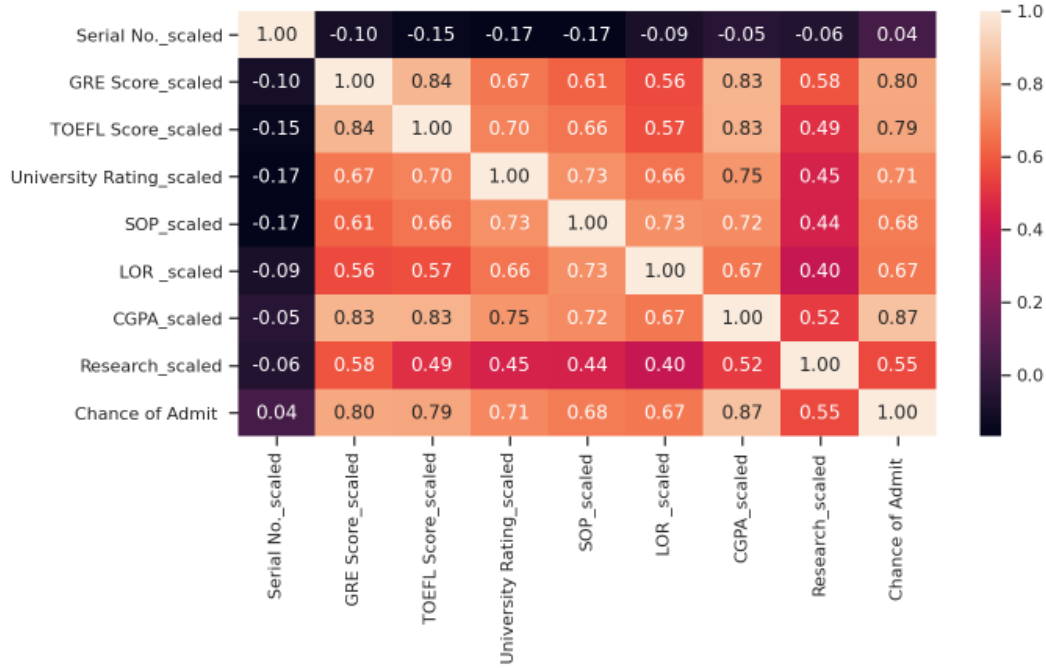
```
[346] fig, ax = plt.subplots(figsize=(10,5))
      sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39162a3828>
```




```
[347] fig, ax = plt.subplots(figsize=(10,5))
      sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f391355e4a8>
```



На основе корреляционной матрицы можно сделать следующие выводы.

1. Корреляционные матрицы для исходных и масштабированных данных совпадают;

2. Колонку «Serial No.» имеет очень слабую корреляцию с целевым признаком, поэтому ее не будем включать в модели.
3. В данном наборе все остальные колонки отлично коррелируют с целевым признаком. Но стоит отметить «Research», его корреляция ниже остальных, но допустика, так что эту колонку стоит оставить.
4. Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

2.5. Выбор метрик для последующей оценки качества моделей

Для оценки качества моделей выберем следующие 3 метрики:

1) Mean absolute error - средняя абсолютная ошибка

2) Mean squared error - средняя квадратичная ошибка

3) Метрика R2 или коэффициент детерминации

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества:

```
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()
```

2.6. Выбор наиболее подходящих моделей для решения задачи

Для задачи регрессии будем использовать следующие модели:

1. Линейная регрессия (LR)
2. Метод ближайших соседей (KNN)
3. Машина опорных векторов(SVR)
4. Решающее дерево (Tree)
5. Случайный лес (RF)
6. Градиентный бустинг (GB)

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

Разделим выборку на обучающую и тестовую:

```
[352] class_cols = ['GRE Score_scaled', 'TOEFL Score_scaled', 'University Rating_scaled', 'SOP_scaled',
            'LOR _scaled', 'CGPA_scaled', 'Research_scaled']
```

```
[353] X = data[class_cols]
      y = data['Chance of Admit ']
      X.shape
```

```
(400, 7)
```

```
[354] # С использованием метода train_test_split разделим выборку на обучающую и тестовую
      regr_X_train, regr_X_test, regr_y_train, regr_y_test = train_test_split(X, y, test_size=0.25, random_state=1)
      print("regr_X_train:", X_train.shape)
      print("regr_X_test:", X_test.shape)
      print("regr_y_train:", y_train.shape)
      print("regr_y_test:", y_test.shape)
```

```
regr_X_train: (300, 8)
regr_X_test: (100, 8)
regr_y_train: (300,)
regr_y_test: (100,)
```

2.8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров

```
[355] # Модели
      regr_models = {'LR': LinearRegression(),
                     'KNN_5':KNeighborsRegressor(n_neighbors=5),
                     'SVR':SVR(),
                     'Tree':DecisionTreeRegressor(),
                     'RF':RandomForestRegressor(),
                     'GB':GradientBoostingRegressor()}
```

```
[356] # Сохранение метрик
      regrMetricLogger = MetricLogger()
```

```
[357] def regr_train_model(model_name, model, regrMetricLogger):
          model.fit(regr_X_train, regr_y_train)
          y_pred = model.predict(regr_X_test)

          mae = mean_absolute_error(regr_y_test, y_pred)
          mse = mean_squared_error(regr_y_test, y_pred)
          r2 = r2_score(regr_y_test, y_pred)

          regrMetricLogger.add('MAE', model_name, mae)
          regrMetricLogger.add('MSE', model_name, mse)
          regrMetricLogger.add('R2', model_name, r2)

          print('*******************************************************')
          print(model)
          print()
          print('MAE={}, MSE={}, R2={}'.format(
              round(mae, 3), round(mse, 3), round(r2, 3)))
          print('*******************************************************')
```

```
for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
******************************************************
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

MAE=0.043, MSE=0.004, R2=0.816
******************************************************
******************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')

MAE=0.053, MSE=0.006, R2=0.737
******************************************************
******************************************************
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE=0.066, MSE=0.007, R2=0.711
******************************************************
******************************************************
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')

MAE=0.07, MSE=0.009, R2=0.616
******************************************************
******************************************************
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)

MAE=0.05, MSE=0.005, R2=0.764
******************************************************
```

```
******************************************************
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)

MAE=0.047, MSE=0.005, R2=0.776
******************************************************
```

2.9. Подбор гиперпараметров для выбранных моделей

```
[360] n_range = np.array(range(1,240,10))
      tuned_parameters = [{'n_neighbors': n_range}]
      tuned_parameters
```

```
[{'n_neighbors': array([  1,  11,  21,  31,  41,  51,  61,  71,  81,  91, 101, 111, 121,
         131, 141, 151, 161, 171, 181, 191, 201, 211, 221, 231])}]
```

```
[361] %%time
      regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5, scoring='neg_mean_squared_error')
      regr_gs.fit(regr_X_train, regr_y_train)
```

```
CPU times: user 521 ms, sys: 924 µs, total: 522 ms
Wall time: 526 ms
```

```
[362] # Лучшая модель
      regr_gs.best_estimator_
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=21, p=2,
                    weights='uniform')
```

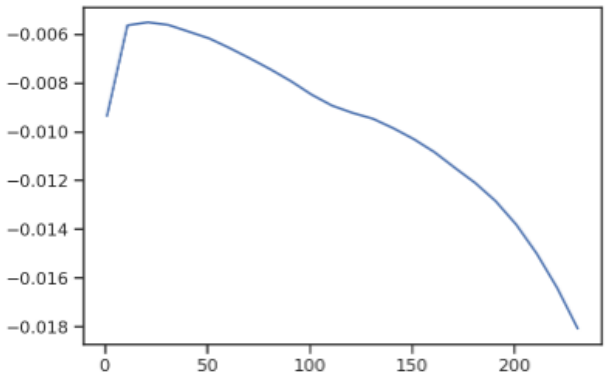
```
[363] # Лучшее значение параметров
      regr_gs.best_params_
```

```
{'n_neighbors': 21}
```

```
[364] # Изменение качества на тестовой выборке в зависимости от К-соседей
      plt.plot(n_range, regr_gs.cv_results_['mean_test_score'])
```


```
[364] # 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 0x7f39131e0a58>]
```



2.10. Построение моделей для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей

```
[365] regr_models_grid = {'KNN_801':regr_gs.best_estimator_}
```

```
[366] for model_name, model in regr_models_grid.items():
          regr_train_model(model_name, model, regrMetricLogger)
```

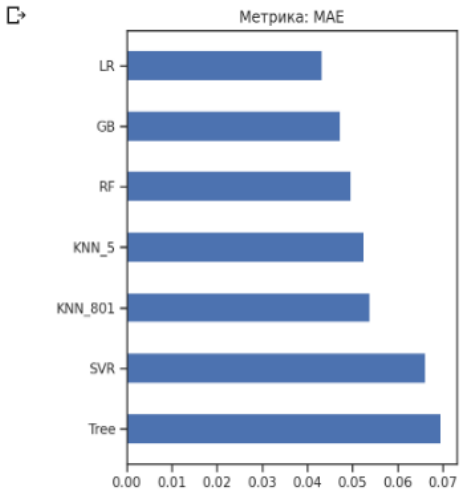
```
**************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=21, p=2,
                    weights='uniform')

MAE=0.054, MSE=0.007, R2=0.712
**************************************************
```

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

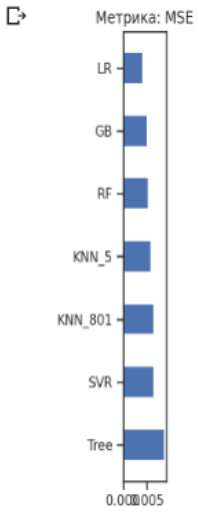
```
[367] # Метрики качества модели
      regr_metrics = regrMetricLogger.df['metric'].unique()
      regr_metrics
```

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

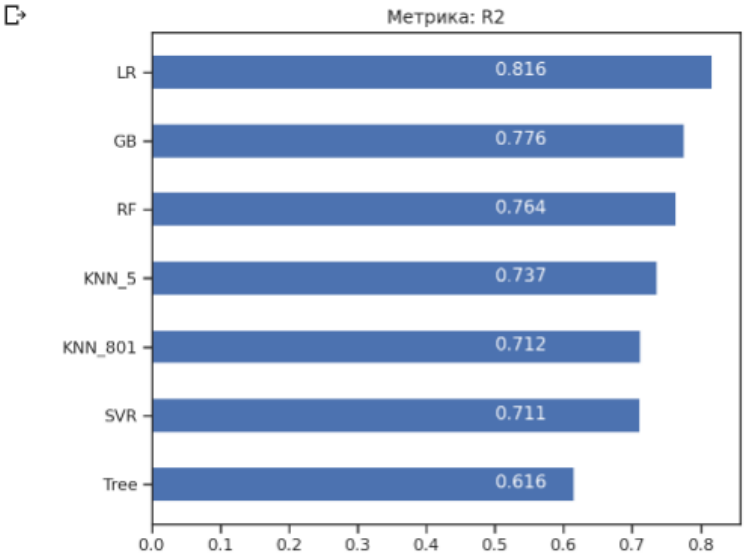
```
[368] regrMetricLogger.plot('Метрика: ' + 'MAE', 'MAE', ascending=False, figsize=(5, 6))
```



```
[369] regrMetricLogger.plot('Метрика: ' + 'MSE', 'MSE', ascending=False, figsize=(0.7, 6))
```




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



Вывод: самой лучшей оказалась модель на основе линейной регрессии.

3. Заключение

В данном курсовом проекте мы выполнили типовую задачу машинного обучения – провести анализ данных, операции над датасетом, подобрать модели, а также наиболее подходящие гиперпараметры выбранных моделей.

В ходе курсового проекта были построены модели, основанные на методах: линейная регрессия, метод ближайших соседей, машина опорных векторов, решающее дерево, случайный лес, градиентный бустинг. В результате сравнения построенных моделей лучше всех себя показала модель на основе линейной регрессии.

Машинное обучение очень актуально в современном мире, оно используется практически во многих сферах. Программист должен подбирать подходящие технологии машинного обучения для достижения наилучших результатов.

4. Список использованных источников

1. Лекции 6-го семестра 2020 года по дисциплине «Технологии машинного обучения»
2. https://scikit-learn.org/stable/index.html
3. https://www.kaggle.com/datasets