Лабораторная работа №5, Грызин Алексей РТ5-61Б

Ансамбли моделей машинного обучения.

- #### Цель лабораторной работы: изучение ансамблей моделей машинного обучения.

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите следующие ансамблевые модели:
   - одну из моделей группы бэггинга (бэггинг или случайный лес или сверхслучайные деревья);
   - одну из моделей группы бустинга;
   - одну из моделей группы стекинга.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

Импорт необходимых библиотек и загрузка набора данных

In [ ]:
```
from operator import itemgetter
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression, RidgeCV
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import BaggingRegressor, RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import graphviz

df = pd.read_csv('data/Admission_Predict.csv')
```

Анализ датасета

The parameters included are :

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

In [ ]:
```
df.head()
```

Out[ ]:

	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

In [ ]:
```
df.describe()
```

Out[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	C
count	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000
mean	200.500000	316.807500	107.410000	3.087500	3.400000	3.452500	8.598
std	115.614301	11.473646	6.069514	1.143728	1.006869	0.898478	0.596
min	1.000000	290.000000	92.000000	1.000000	1.000000	1.000000	6.800
25%	100.750000	308.000000	103.000000	2.000000	2.500000	3.000000	8.170
50%	200.500000	317.000000	107.000000	3.000000	3.500000	3.500000	8.610
75%	300.250000	325.000000	112.000000	4.000000	4.000000	4.000000	9.062
max	400.000000	340.000000	120.000000	5.000000	5.000000	5.000000	9.920

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Serial No.         400 non-null    int64  
 1   GRE Score          400 non-null    int64  
 2   TOEFL Score        400 non-null    int64  
 3   University Rating  400 non-null    int64  
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64  
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

In [ ]:
```
df.isnull().sum()
```

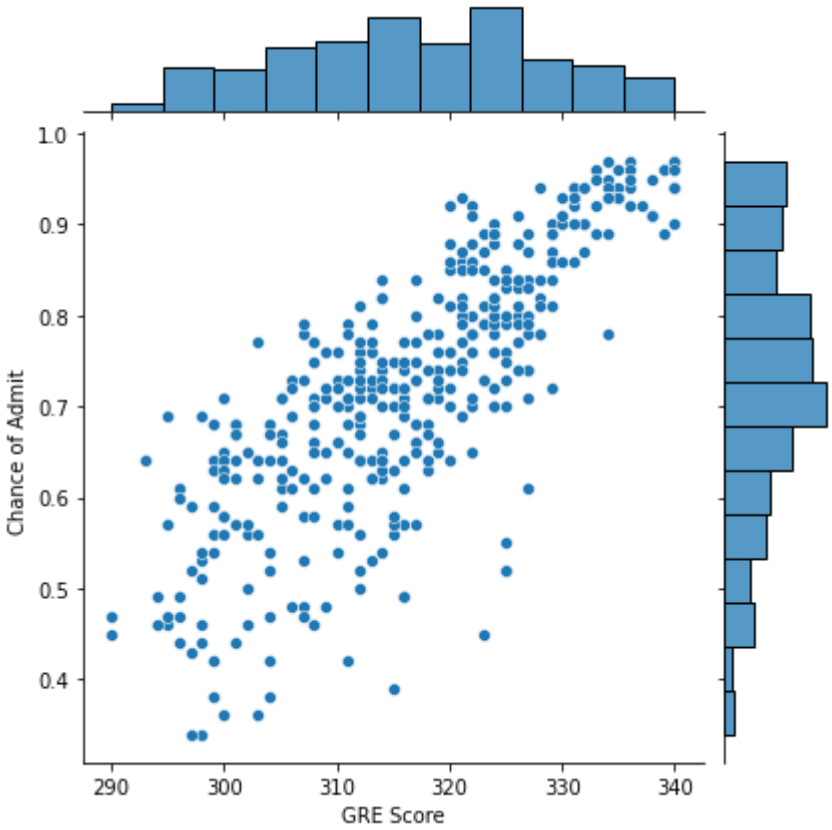
Out[ ]:
```
Serial No.           0
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

> Как видно, пропуски отсутствуют, а значит нет необходимости в удалении колонок или строк.

Диаграмма Jointplot

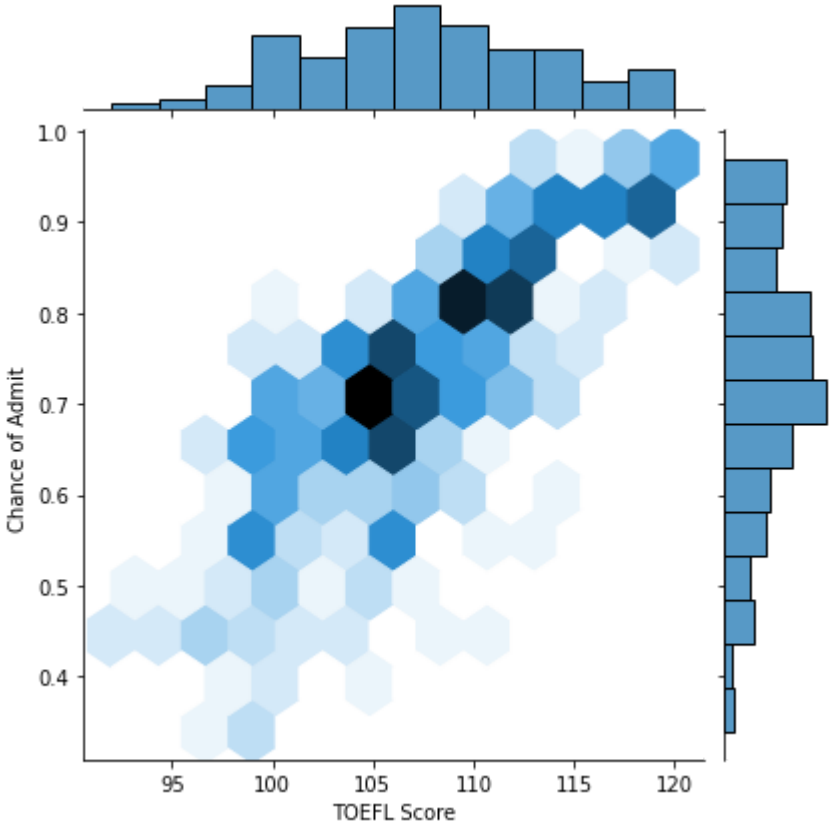
In [ ]:
```
sns.jointplot(x="GRE Score", y="Chance of Admit ", data=df)
```

Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x28eeedf70>
```



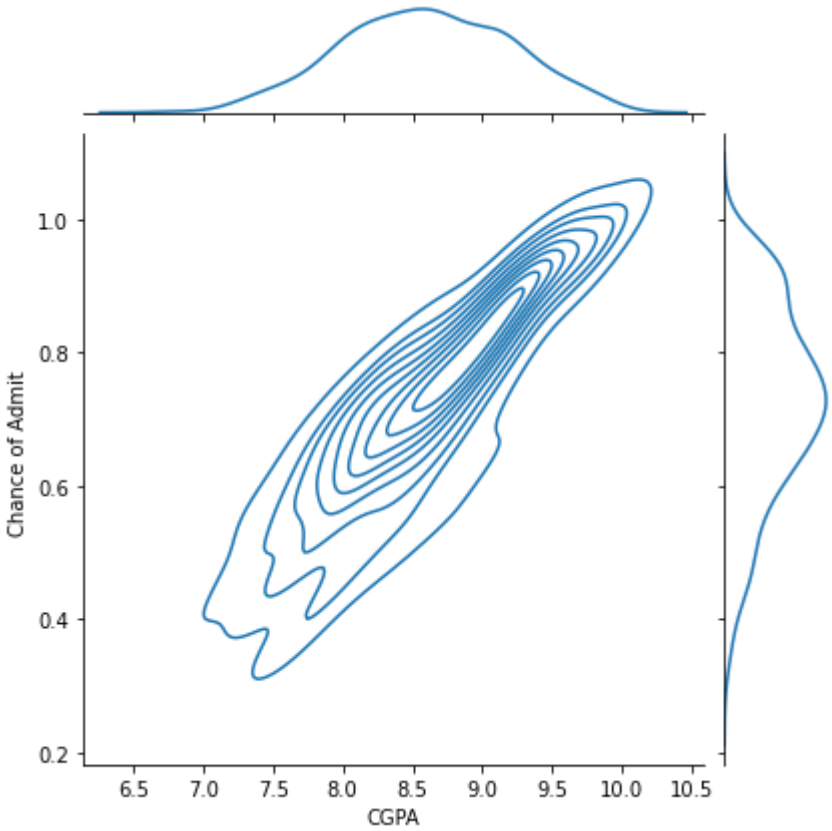
In [ ]:
```
sns.jointplot(x="TOEFL Score", y="Chance of Admit ", data=df, kind='hex')
```

Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x29008e5e0>
```



In [ ]:
```
sns.jointplot(x="CGPA", y="Chance of Admit ", data=df, kind="kde")
```

Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x290171ca0>
```



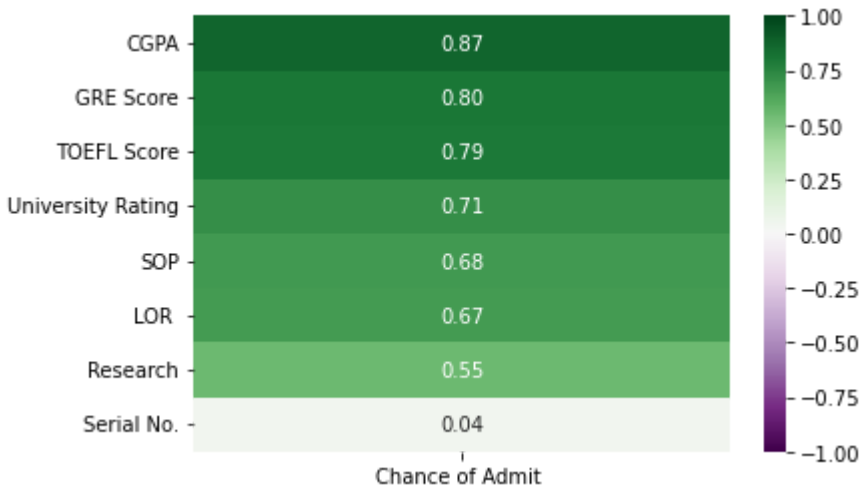
Корреляционный анализ

- В данном датасете целевым признаком является параметр "Chance of Admit".

Рассмотрим, как остальные параметры с ним коррелируют.

In [ ]:
```
chance_of_admit = pd.DataFrame(df.corr()["Chance of Admit "].sort_values(asc

sns.heatmap(chance_of_admit, annot=True,  fmt='.2f', cmap=plt.cm.PRGn, vmin=
```

Out[ ]:
```
<AxesSubplot:>
```



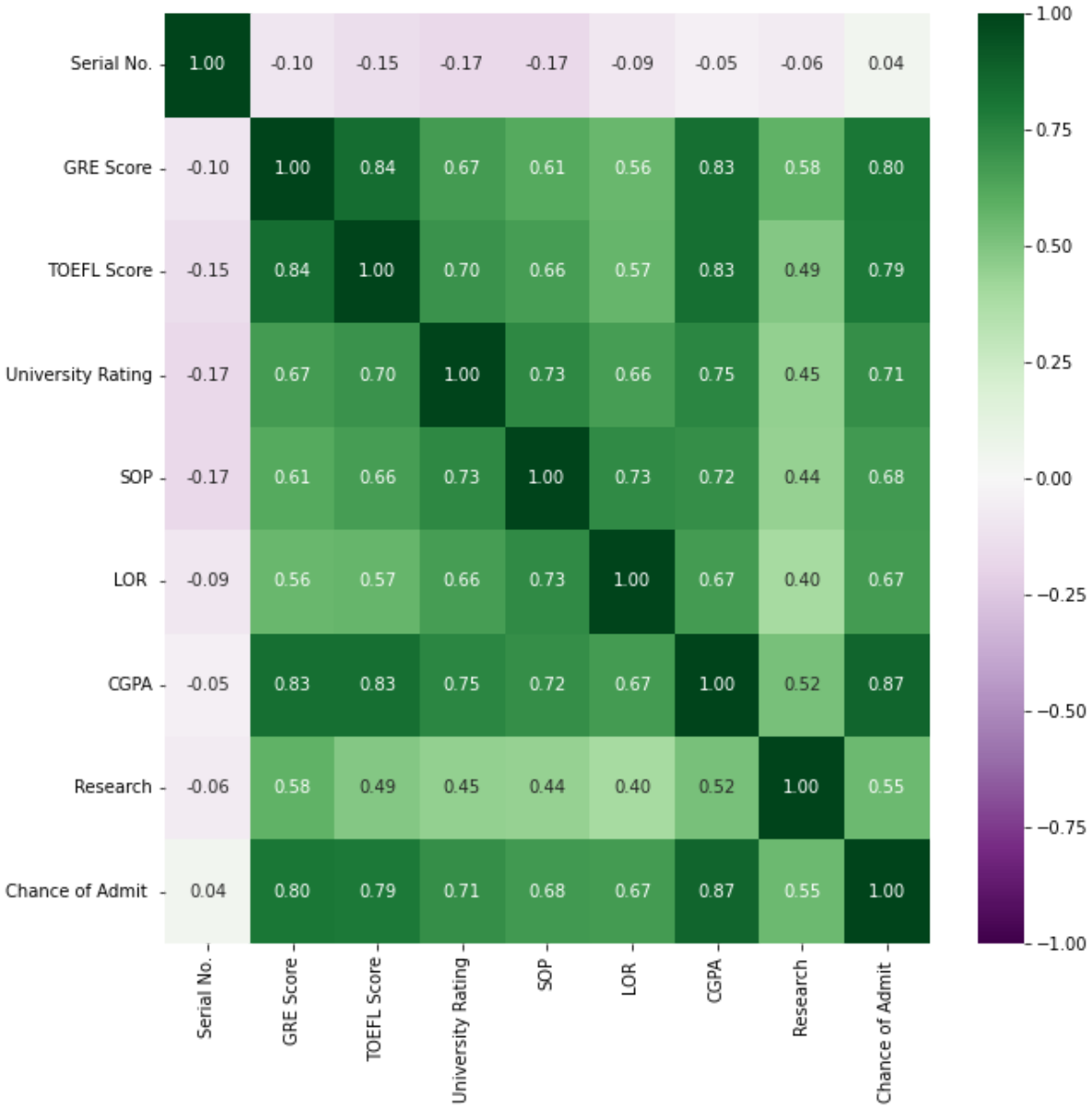
In [ ]:
```
df.corr()
```

Out[ ]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	R
Serial No.	1.000000	-0.097526	-0.147932	-0.169948	-0.166932	-0.088221	-0.045608	-(
GRE Score	-0.097526	1.000000	0.835977	0.668976	0.612831	0.557555	0.833060	(
TOEFL Score	-0.147932	0.835977	1.000000	0.695590	0.657981	0.567721	0.828417	0
University Rating	-0.169948	0.668976	0.695590	1.000000	0.734523	0.660123	0.746479	(
SOP	-0.166932	0.612831	0.657981	0.734523	1.000000	0.729593	0.718144	0
LOR	-0.088221	0.557555	0.567721	0.660123	0.729593	1.000000	0.670211	0
CGPA	-0.045608	0.833060	0.828417	0.746479	0.718144	0.670211	1.000000	(
Research	-0.063138	0.580391	0.489858	0.447783	0.444029	0.396859	0.521654	1
Chance of Admit	0.042336	0.802610	0.791594	0.711250	0.675732	0.669889	0.873289	(

In [ ]:
```
fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(10, 10))

sns.heatmap(df.corr(), annot=True,  fmt='.2f', cmap=plt.cm.PRGn, vmin=-1, vm
```

Out[ ]:
```
<AxesSubplot:>
```



Удаление лишних столбцов и масштабирование данных

In [ ]:
```
# Удаление лишних колонок
df_corr = df.drop(columns=["Serial No.", "TOEFL Score", "Research", "LOR ",
df_corr.head()
```

Out[ ]:

	GRE Score	University Rating	CGPA	Chance of Admit
0	337	4	9.65	0.92
1	324	4	8.87	0.76
2	316	3	8.00	0.72
3	322	3	8.67	0.80
4	314	2	8.21	0.65

In [ ]:
```
# Масштабирование
scaler = MinMaxScaler()
scaled_features = scaler.fit_transform(X=df_corr.drop(columns=["Chance of Ad

# Вставка отмасштабированных данных
df_scaled = pd.DataFrame(scaled_features, columns=df_corr.columns[:-1])
df_scaled.head()
```

Out[ ]:

	GRE Score	University Rating	CGPA
0	0.94	0.75	0.913462
1	0.68	0.75	0.663462
2	0.52	0.50	0.384615
3	0.64	0.50	0.599359
4	0.48	0.25	0.451923

Разбиение выборки на обучающую и тестовую

In [ ]:
```
x = df_scaled
y = df_corr["Chance of Admit "]

x_train: pd.DataFrame
x_test: pd.DataFrame
y_train: pd.Series
y_test: pd.Series

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4, ran
```

In [ ]:
```
# Размер обучающей выборки
x_train.shape, y_train.shape
```

Out[ ]:
```
((240, 3), (240,))
```

In [ ]:
```
# Размер тестовой выборки
x_test.shape, y_test.shape
```

Out[ ]:
```
((160, 3), (160,))
```

Обучение моделей

In [ ]:
```
# Список моделей и списки метрик
model_list =[]
mae_list = []
mse_list = []
r2_list = []
```

In [ ]:
```
def plot_feature_importances(feature_names, tree_model):
    """
    Функция визуализации важности признаков

    :param feature_names: Названия признаков
    :param tree_model: Модель
    """
    feature_importance_list = list(zip(feature_names, tree_model.feature_imp
    sorted_list = sorted(feature_importance_list, key=itemgetter(1), reverse
    feature_order = [x for x, _ in sorted_list]

    plt.figure(figsize=(9, 6))
    bar_plot = sns.barplot(x=feature_names, y=tree_model.feature_importances
    bar_plot.bar_label(bar_plot.containers[-1], fmt="%.3f")
    plt.show()
```

In [ ]:
```
def generate_tree_graph(estimator, filename):
    """
```

```
    Функция для создания графа дерева

    :param estimator: Модель
    :param filename: Название файла с графом
    """
    dot_data = export_graphviz(
        estimator, feature_names=list(x_train.columns), filled=True, rounded
    )
    graph = graphviz.Source(dot_data, format="svg", directory="images", file
    graph.render()
```

Бэггинг

In [ ]:
```
# Обучение модели
bg_regressor = BaggingRegressor(n_estimators=100, oob_score=True, random_sta
bg_regressor.fit(x_train, y_train)
```

Out[ ]:
```
BaggingRegressor(n_estimators=100, oob_score=True, random_state=1)
```

In [ ]:
```
# Out-of-bag error, возвращаемый регрессором
# Для регресии используется метрика r2
bg_regressor.oob_score_
```

Out[ ]:
```
0.7370326986644662
```

In [ ]:
```
# Предсказание, сделанное моделью
bg_pred = bg_regressor.predict(x_test)
```

In [ ]:
```
# Запись модели и значений метрик в соответствующий списки
model_list.append('Bagging')
mae_list.append(mean_absolute_error(y_test, bg_pred))
mse_list.append(mean_squared_error(y_test, bg_pred))
r2_list.append(r2_score(y_test, bg_pred))
```

Случайный лес

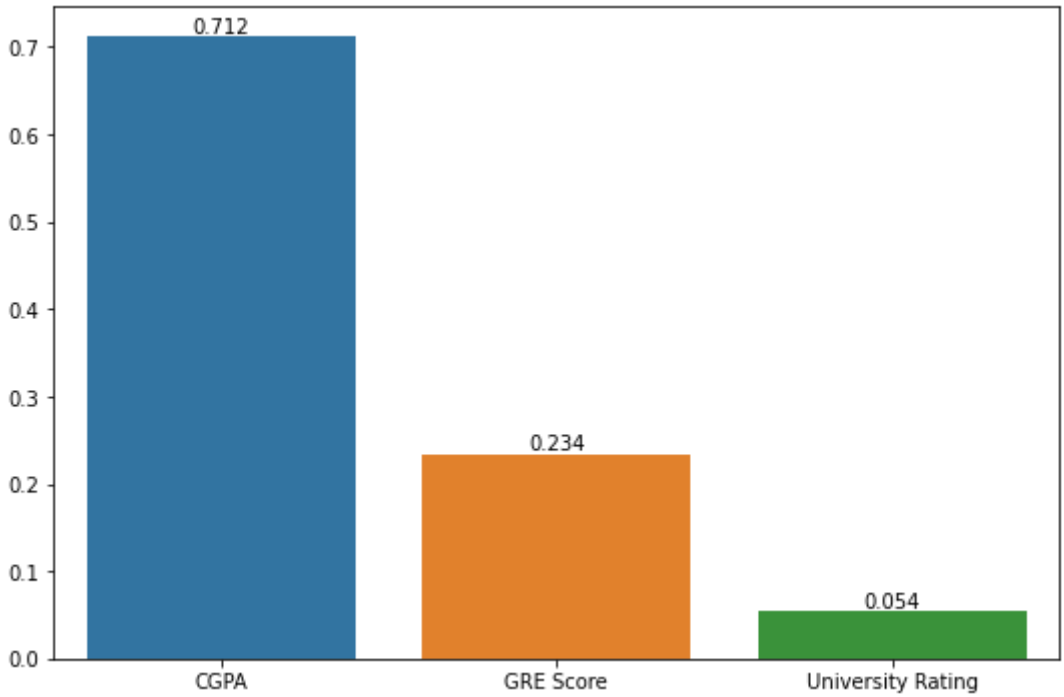
In [ ]:
```
# Обучение модели
rf_regressor = RandomForestRegressor(n_estimators=100, criterion='absolute_e
rf_regressor.fit(x_train, y_train)
```

Out[ ]:
```
RandomForestRegressor(criterion='absolute_error', oob_score=True,
                      random_state=1)
```

In [ ]:
```
# Out-of-bag error, возвращаемый регрессором
rf_regressor.oob_score_
```

Out[ ]:
```
0.7402371832366637
```

In [ ]:
```
# Важность признаков случайного леса
plot_feature_importances(x.columns.values, rf_regressor)
```



In [ ]:
```
# Предсказание, сделанное моделью
rf_pred = rf_regressor.predict(x_test)
```

In [ ]:
```
# Запись модели и значений метрик в соответствующий списки
model_list.append('Random Forest')
mae_list.append(mean_absolute_error(y_test, rf_pred))
mse_list.append(mean_squared_error(y_test, rf_pred))
r2_list.append(r2_score(y_test, rf_pred))
```

Бустинг

In [ ]:
```
# Градиентный бустинг
# Гиперпараметры для оптимизации
parameters_to_tune = {
    "n_estimators": [5, 10, 15],
    "learning_rate": np.linspace(0.1, 0.3, 3),
    "min_samples_split": np.arange(2, 5, 1),
    "max_depth": np.arange(1, 10, 1),
}
```

In [ ]:
```
%%time
# Оптимизация гиперпараметров
gb_gs = GridSearchCV(GradientBoostingRegressor(random_state=3),
                     parameters_to_tune, cv=4, scoring='neg_root_mean_squar
gb_gs.fit(x_train, y_train)
```

```
CPU times: user 4.08 s, sys: 22.3 ms, total: 4.1 s
Wall time: 4.1 s
```

Out[ ]:
```
GridSearchCV(cv=4, estimator=GradientBoostingRegressor(random_state=3),
             param_grid={'learning_rate': array([0.1, 0.2, 0.3]),
                         'max_depth': array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
                         'min_samples_split': array([2, 3, 4]),
                         'n_estimators': [5, 10, 15]},
             scoring='neg_root_mean_squared_error')
```

In [ ]:
```
# Лучшее значение параметров
gb_gs.best_params_
```

Out[ ]:
```
{'learning_rate': 0.3,
 'max_depth': 1,
 'min_samples_split': 2,
 'n_estimators': 15}
```

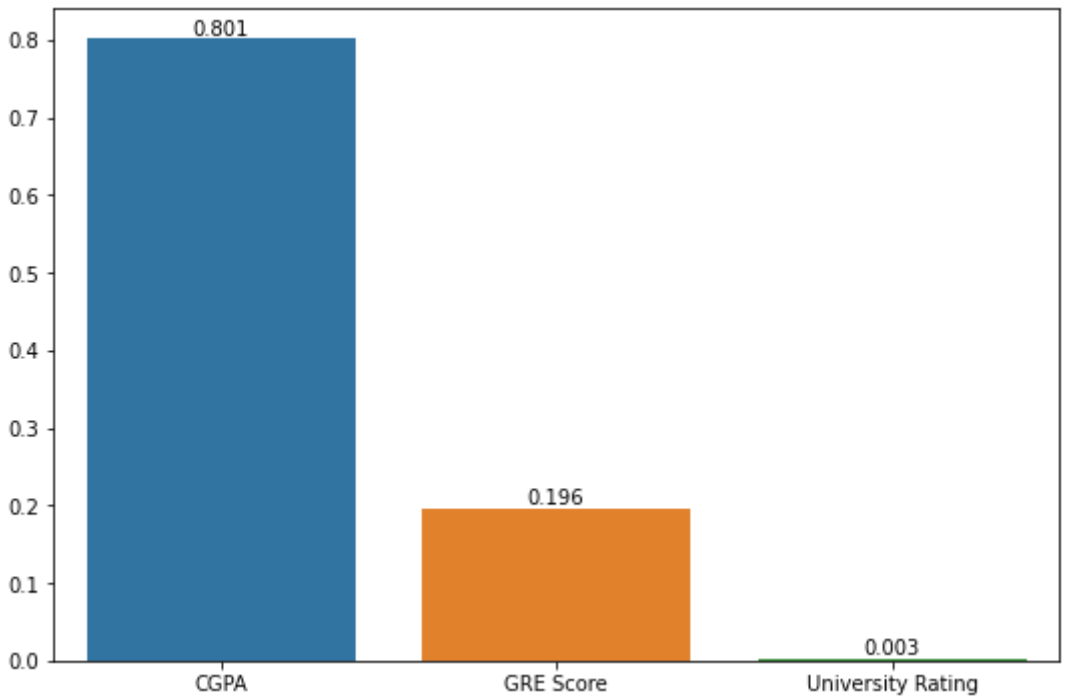
In [ ]:
```
# Лучшее значение метрики
gb_gs.best_score_
```

Out[ ]:
```
-0.06480835067633028
```

In [ ]:
```
# Обучение модели
gb_regressor : GradientBoostingRegressor = gb_gs.best_estimator_
gb_regressor.fit(x_train, y_train)
```

Out[ ]:
```
GradientBoostingRegressor(learning_rate=0.3, max_depth=1, n_estimators=15,
                          random_state=3)
```

In [ ]:
```
# Важность признаков градиентного бустинга
plot_feature_importances(x.columns.values, gb_regressor)
```



In [ ]:
```
# Предсказание, сделанное моделью
gb_pred = gb_regressor.predict(x_test)
```

In [ ]:
```
# Запись модели и значений метрик в соответствующий списки
model_list.append('Gradient Boosting')
mae_list.append(mean_absolute_error(y_test, gb_pred))
mse_list.append(mean_squared_error(y_test, gb_pred))
r2_list.append(r2_score(y_test, gb_pred))
```

Стекинг

In [ ]:
```
# Полиномиальная регрессия
pr_pipeline = make_pipeline(MinMaxScaler(), PolynomialFeatures(degree=2), Li
```

In [ ]:
```
# Дерево решений
dt_pipeline = DecisionTreeRegressor(
    criterion="absolute_error", max_depth=7, max_features='auto', min_sample
)
```

In [ ]:
```
# Значения коэффициента альфа
alphas=[0.001, 0.01, 0.1, 1]
```

In [ ]:
```
# Обучение модели
estimators = [
    ("Polynomial Regression", pr_pipeline),
    ("Decision Tree", dt_pipeline),
    ("Gradient Boosting", gb_regressor),
    ("Bagging", bg_regressor)
]

stacking_regressor = StackingRegressor(estimators=estimators, final_estimato
stacking_regressor.fit(x_train, y_train)
```

Out[ ]:
```
StackingRegressor(estimators=[('Polynomial Regression',
                               Pipeline(steps=[('minmaxscaler', MinMaxScaler
()),
                                               ('polynomialfeatures',
                                                PolynomialFeatures()),
                                               ('linearregression',
                                                LinearRegression())])),
                              ('Decision Tree',
                               DecisionTreeRegressor(criterion='absolute_err
or',
                                                     max_depth=7,
                                                     max_features='auto',
                                                     min_samples_leaf=0.03,
                                                     random_state=8)),
                              ('Gradient Boosting',
                               GradientBoostingRegressor(learning_rate=0.3,
                                                         max_depth=1,
                                                         n_estimators=15,
                                                         random_state=3)),
                              ('Bagging',
                               BaggingRegressor(n_estimators=100,
                                                oob_score=True,
                                                random_state=1))],
                  final_estimator=RidgeCV(alphas=array([0.001, 0.01 , 0.1  ,
1.   ])))
```

In [ ]:
```
# Предсказание, сделанное моделью
st_pred = stacking_regressor.predict(x_test)
```

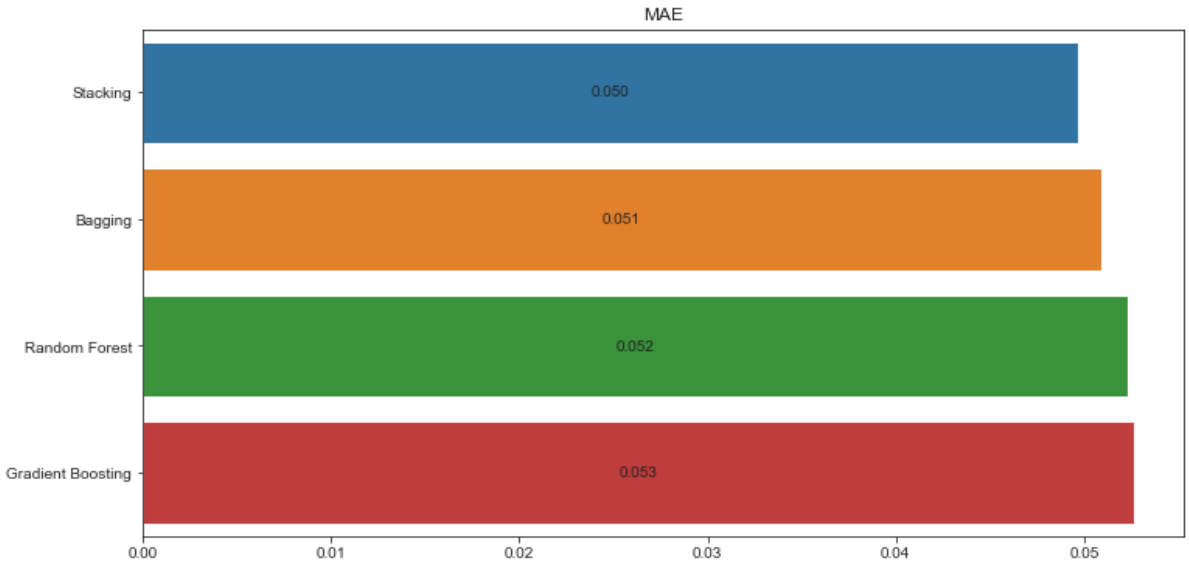
In [ ]:
```
# Запись модели и значений метрик в соответствующий списки
model_list.append('Stacking')
mae_list.append(mean_absolute_error(y_test, st_pred))
mse_list.append(mean_squared_error(y_test, st_pred))
r2_list.append(r2_score(y_test, st_pred))
```

Оценка качества моделей

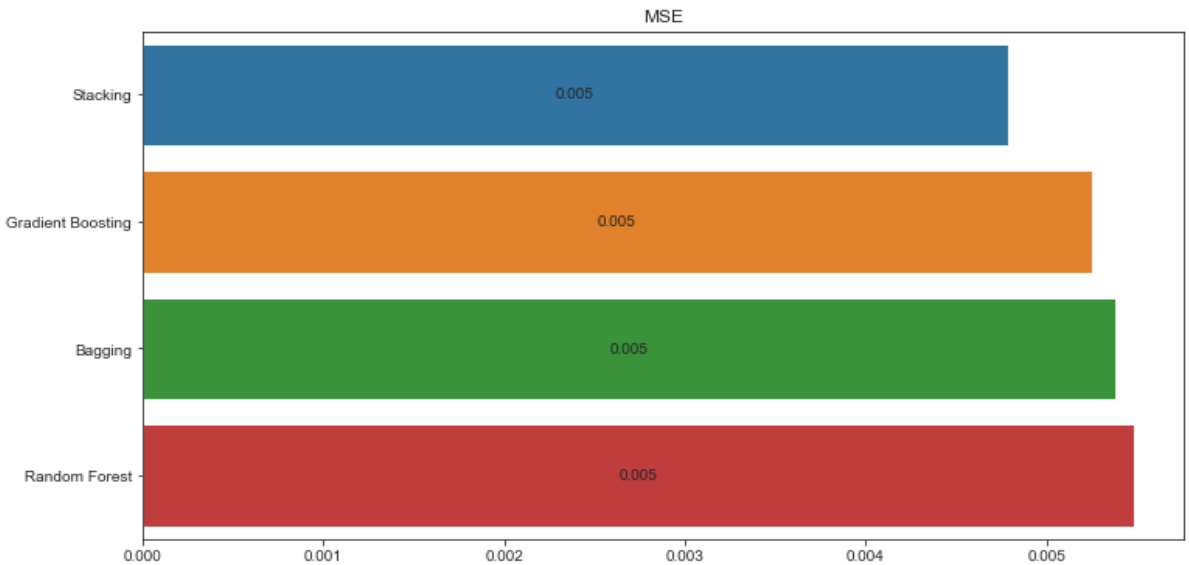
In [ ]:
```
def plot_metrics(models_list, metrics_list, metric_name):
    """
    Функция визуализации метрик

    :param models_list: Список моделей
    :param metrics_list: Список метрик
    :param metric_name: Название метрики
    """
    model_metric_list = list(zip(models_list, metrics_list))
    sorted_list = sorted(model_metric_list, key=itemgetter(1))
    feature_order = [x for x, _ in sorted_list]
```

```
    with sns.axes_style("ticks"):
        plt.figure(figsize=(12, 6))
        bar_plot = sns.barplot(x=metrics_list, y=models_list, order=feature_
        bar_plot.bar_label(bar_plot.containers[-1], label_type="center", fmt
        plt.title(metric_name)
        plt.show()
```

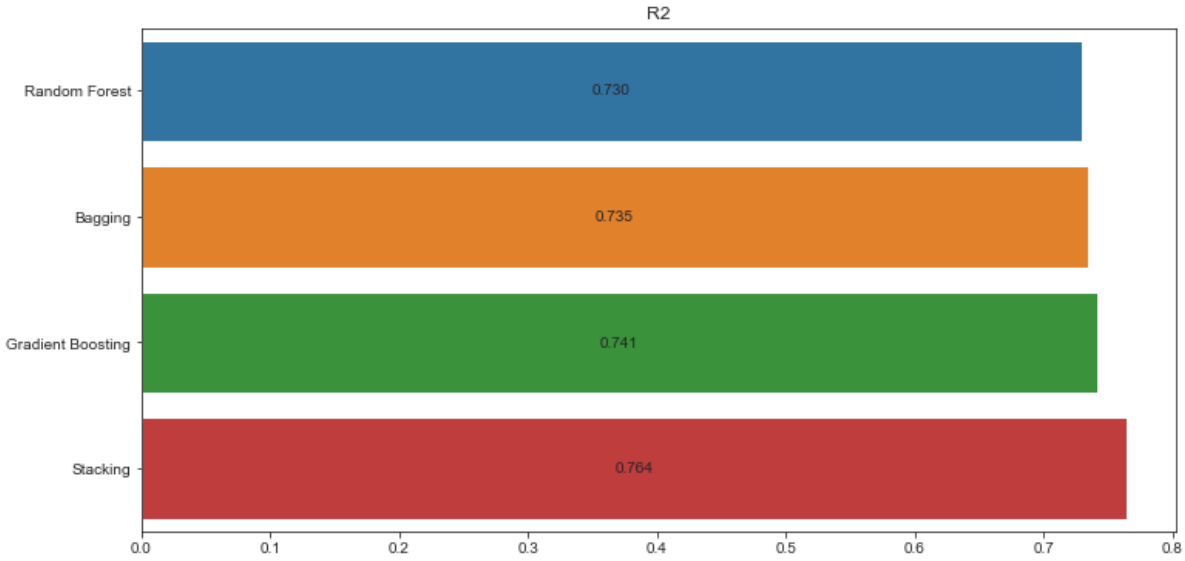
In [ ]:
```
# Сравнение MAE
plot_metrics(model_list, mae_list, 'MAE')
```



In [ ]:
```
# Сравнение MSE
plot_metrics(model_list, mse_list, 'MSE')
```



In [ ]:
```
# Сравнение R2
plot_metrics(model_list, r2_list, 'R2')
```


R2
Random Forest
Bagging
Gradient Boosting
Stacking
0.730
0.735
0.741
0.764
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8