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)»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по курсу
«Data Science»

Тема: «Прогнозирование конечных свойств новых материалов (композиционных материалов)»

Слушатель Величко Ольга Андреевна

Москва, 2022

Задачи работы

- Разработать и обучить несколько моделей регрессии для прогноза целевых показателей нового композиционного материала
- Оформить получение пользователем целевого показателя в веб-приложение
- Целевые показатели - модуль упругости при растяжении и прочность при растяжении, соотношение «матрица-наполнитель»

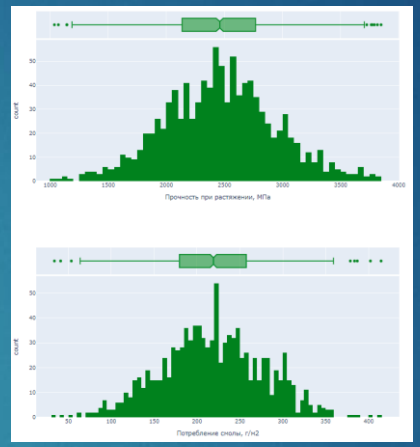
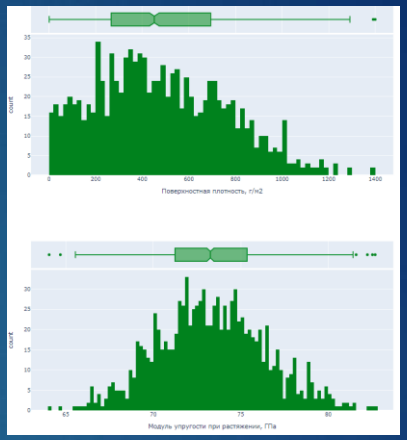
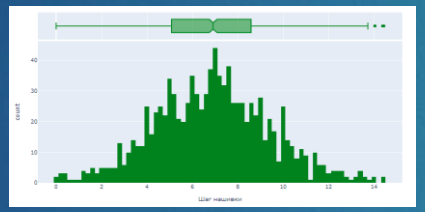
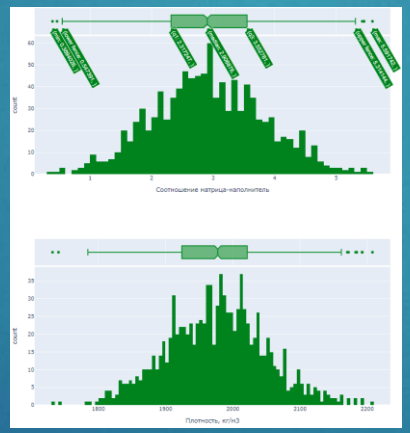
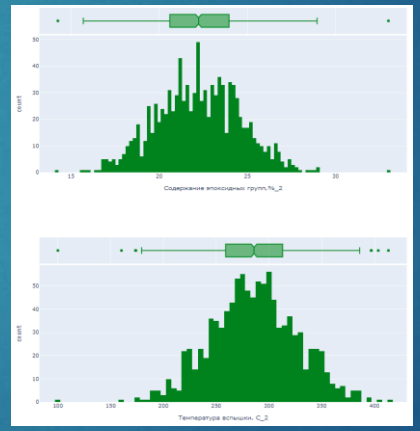
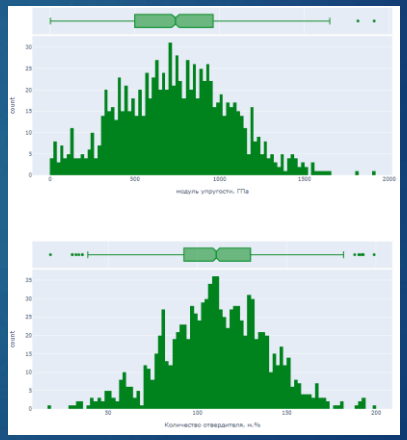
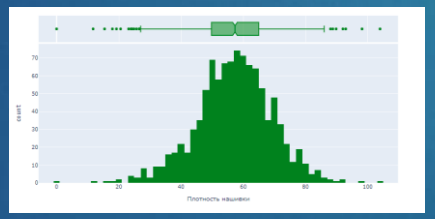
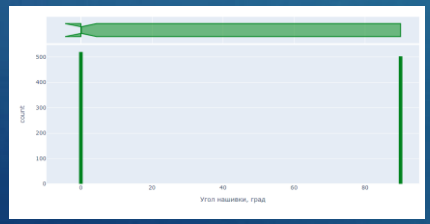
Знакомство с данными

- 13 признаков по 1023 значения
- Пропуски и дубликаты отсутствуют
- Тип данных большинства переменных – вещественный, переменные непрерывны
- Распределение данных близко к нормальному
- Линейной зависимости между признаками нет
- Слабая корреляция переменных между собой

Данные до предобработки

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град
0	1.857143	2030.000000	738.736842	30.000000	22.267857	100.000000	210.000000	70.000000	3000.000000	220.000000	0
1	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385	210.000000	70.000000	3000.000000	220.000000	0
2	1.857143	2030.000000	738.736842	49.900000	33.000000	284.615385	210.000000	70.000000	3000.000000	220.000000	0
3	1.857143	2030.000000	738.736842	129.000000	21.250000	300.000000	210.000000	70.000000	3000.000000	220.000000	0
4	2.771331	2030.000000	753.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0
...	...	...	...	...	...	...	...	...	...	...	...
1018	2.271346	1952.087902	912.855545	86.992183	20.123249	324.774576	209.198700	73.090961	2387.292495	125.007669	90
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401	350.660830	72.920827	2360.392784	117.730099	90
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047	740.142791	74.734344	2662.906040	236.606764	90
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840	641.468152	74.042708	2071.715856	197.126067	90
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708	758.747882	74.309704	2856.328932	194.754342	90

1023 rows × 13 columns

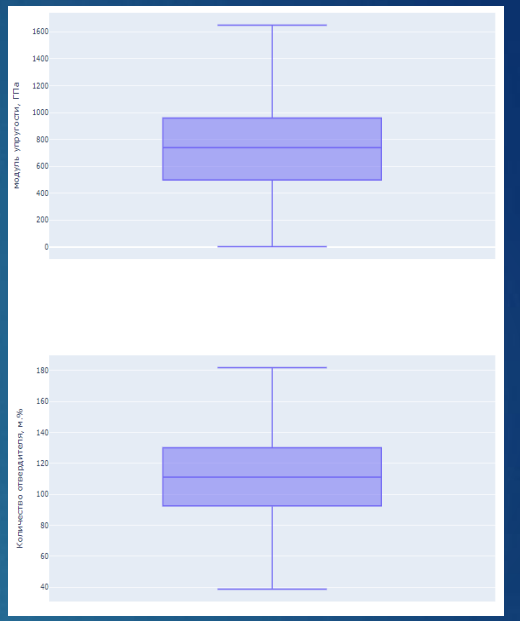









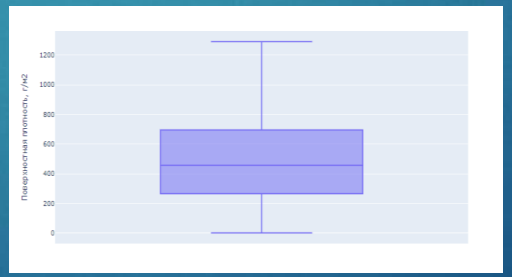
Данные после удаления выбросов

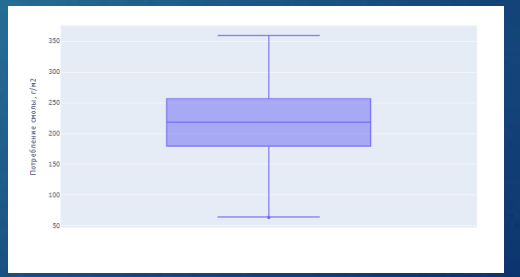
```
# Удалим выбросы далее 1,5 межквартильных размахов

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3-Q1

df = df[~((df<(Q1-1.5*IQR))|(df>(Q3+1.5*IQR))).any(axis=1)]
df
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град
1	1.857143	2030.000000	738.736842	50.000000	23.750000	284.615385	210.000000	70.000000	3000.000000	220.000000	0
3	1.857143	2030.000000	738.736842	129.000000	21.250000	300.000000	210.000000	70.000000	3000.000000	220.000000	0
4	2.771331	2030.000000	753.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0
5	2.767918	2000.000000	748.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0
6	2.569620	1910.000000	807.000000	111.860000	22.267857	284.615385	210.000000	70.000000	3000.000000	220.000000	0
...	...	...	...	...	...	...	...	...	...	...	...
1018	2.271346	1952.087902	912.855545	86.992183	20.123249	324.774576	209.198700	73.090961	2387.292495	125.007669	90
1019	3.444022	2050.089171	444.732634	145.981978	19.599769	254.215401	350.660830	72.920827	2360.392784	117.730099	90
1020	3.280604	1972.372865	416.836524	110.533477	23.957502	248.423047	740.142791	74.734344	2662.906040	236.606764	90
1021	3.705351	2066.799773	741.475517	141.397963	19.246945	275.779840	641.468152	74.042708	2071.715856	197.126067	90
1022	3.808020	1890.413468	417.316232	129.183416	27.474763	300.952708	758.747882	74.309704	2856.328932	194.754342	90

936 rows × 13 columns







Корреляция переменных

- Корреляция слабая. Max 0,0059.

```
# Корреляционная матрица

fig = px.imshow(df.corr(), text_auto=True, width=1000, height=1000, aspect="auto", color_continuous_scale='thermal')
fig.update_xaxes(side="top", tickangle=90)
fig.show()
```

Нормализация данных и предобработка перед обучением

```
# Нормализация

scaler = preprocessing.MinMaxScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df_normalized
```

	Соотношение матрица-наполнитель	Плотность, кг/м3	модуль упругости, ГПа	Количество отвердителя, м.%	Содержание эпоксидных групп,%_2	Температура вспышки, С_2	Поверхностная плотность, г/м2	Модуль упругости при растяжении, ГПа	Прочность при растяжении, МПа	Потребление смолы, г/м2	Угол нашивки, град
0	0.274768	0.651097	0.447061	0.079153	0.607435	0.509164	0.162230	0.280303	0.712590	0.529221	0.0
1	0.274768	0.651097	0.447061	0.630983	0.418887	0.583596	0.162230	0.280303	0.712590	0.529221	0.0
2	0.466552	0.651097	0.455721	0.511257	0.495653	0.509164	0.162230	0.280303	0.712590	0.529221	0.0
3	0.465836	0.571539	0.452685	0.511257	0.495653	0.509164	0.162230	0.280303	0.712590	0.529221	0.0
4	0.424236	0.332865	0.488508	0.511257	0.495653	0.509164	0.162230	0.280303	0.712590	0.529221	0.0
...	...	...	...	...	...	...	...	...	...	...	...
931	0.361662	0.444480	0.552781	0.337550	0.333908	0.703458	0.161609	0.475147	0.463043	0.207613	1.0
932	0.607674	0.704373	0.268550	0.749605	0.294428	0.362087	0.271207	0.464422	0.452087	0.182974	1.0
933	0.573391	0.498274	0.251612	0.501991	0.623085	0.334063	0.572959	0.578740	0.575296	0.585446	1.0
934	0.662497	0.748688	0.448724	0.717585	0.267818	0.466417	0.496511	0.535142	0.334513	0.451779	1.0
935	0.684036	0.280923	0.251903	0.632264	0.888354	0.588206	0.587373	0.551972	0.654075	0.443749	1.0

936 rows × 13 columns

```
# Определим входы и выходы для моделей

target_var_1 = df_normalized['Модуль упругости при растяжении, ГПа']
target_var_2 = df_normalized['Прочность при растяжении, МПа']

train_vars_1 = df_normalized.loc[:, df_normalized.columns != 'Модуль упругости при растяжении, ГПа']
train_vars_2 = df_normalized.loc[:, df_normalized.columns != 'Прочность при растяжении, МПа']
```

```
# Разбиваем выборки на обучающую и тестовую

x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(train_vars_1, target_var_1, test_size=0.3, random_state=14)
x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(train_vars_2, target_var_2, test_size=0.3, random_state=14)
```

```
y_train_1
```

```
y_test_1
```

```
# Проверка правильности разбивки

df_normalized.shape[0] - x_train_1.shape[0] - x_test_1.shape[0]
df_normalized.shape[0] - x_train_2.shape[0] - x_test_2.shape[0]
```

```
0
```

```
# Будем складывать названия моделей и их ошибки в датафрейм
df_errors = pd.DataFrame(columns=['target_var','model_name','MSE','R2'])
```

```
# Зададим функцию для визуализации факт/прогноз для результатов моделей

def result_plot(orig, predict, var, model_name):
    plt.figure(figsize=(17,5))
    plt.title(f'Тестовые и прогнозные значения: {model_name}')
    plt.plot(orig, label='Тест')
    plt.plot(predict, label='Прогноз')
    plt.legend(loc='best')
    plt.ylabel(var)
    plt.xlabel('Количество наблюдений')
    plt.show()
```

Для выравнивания диапазонов входных переменных проведем нормализацию. Так мы облегчим работу моделям.

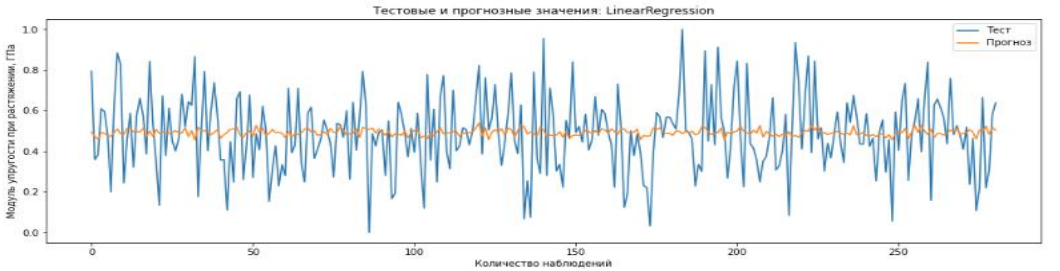
Определим тестовый и обучающий наборы данных

Линейная регрессия

```python
# Линейная регрессия для модуля упругости при растяжении

linear_reg_1 = LinearRegression()
linear_reg_1.fit(x_train_1, y_train_1)
prediction_y_test_linear_1 = linear_reg_1.predict(x_test_1)

# The mean squared error: MSE of zero (0) represents the fact that the predictor is a perfect predictor.
MSE_1 = mean_squared_error(y_test_1, prediction_y_test_linear_1)
# The coefficient of determination: 1 is perfect prediction
R2_1 = r2_score(y_test_1, prediction_y_test_linear_1)

# Линейная регрессия для прочности при растяжении
linear_reg_2 = LinearRegression()
linear_reg_2.fit(x_train_2, y_train_2)
prediction_y_test_linear_2 = linear_reg_2.predict(x_test_2)

MSE_2 = mean_squared_error(y_test_2, prediction_y_test_linear_2)
R2_2 = r2_score(y_test_2, prediction_y_test_linear_2)


linear_errors = pd.DataFrame({'model_name':'LinearRegression',\
                              'target_var':['Модуль упругости при растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                             'MSE':[MSE_1, MSE_2],\
                             'R2':[R2_1, R2_2]})
df_errors = pd.concat([df_errors, linear_errors], ignore_index=True)
df_errors
```

	target_var	model_name	MSE	R2
0	Модуль упругости при растяжении, ГПа	LinearRegression	0.035136	0.014854
1	Прочность при растяжении, МПа	LinearRegression	0.033425	-0.030041

```python
# Отрисуем результат

result_plot(y_test_1.values, prediction_y_test_linear_1, 'Модуль упругости при растяжении, ГПа', 'LinearRegression')
result_plot(y_test_2.values, prediction_y_test_linear_2, 'Прочность при растяжении, МПа', 'LinearRegression')
```



Регрессия k-ближайших соседей

```
# Регрессия k-ближайших соседей
```

```
%%time

neigh = KNeighborsRegressor()
neigh_params = {'n_neighbors' : range(1, 101, 1),
          'weights' : ['uniform', 'distance'],
          'algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute']
          }

GSCV_neigh_1 = GridSearchCV(neigh, neigh_params, n_jobs=-1, cv=10)
GSCV_neigh_1.fit(x_train_1, y_train_1)
# GSCV_neigh_1.best_params_
neigh_1 = GSCV_neigh_1.best_estimator_

MSE_1 = mean_squared_error(y_test_1, neigh_1.predict(x_test_1))
R2_1 = r2_score(y_test_1, neigh_1.predict(x_test_1))

GSCV_neigh_2 = GridSearchCV(neigh, neigh_params, n_jobs=-1, cv=10)
GSCV_neigh_2.fit(x_train_2, y_train_2)
# GSCV_neigh_2.best_params_
neigh_2 = GSCV_neigh_2.best_estimator_

MSE_2 = mean_squared_error(y_test_2, neigh_2.predict(x_test_2))
R2_2 = r2_score(y_test_2, neigh_2.predict(x_test_2))


neigh_errors = pd.DataFrame({'model_name':'KNeighborsRegressor',\
                             'target_var':['Модуль упругости при растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                            'MSE':[MSE_1, MSE_2],\
                            'R2':[R2_1, R2_2]})
df_errors = pd.concat([df_errors, neigh_errors], ignore_index=True)
df_errors
```

	target_var	model_name	MSE	R2
0	Модуль упругости при растяжении, ГПа	LinearRegression	0.035136	0.014854
1	Прочность при растяжении, МПа	LinearRegression	0.033425	-0.030041
2	Модуль упругости при растяжении, ГПа	KNeighborsRegressor	0.035057	0.017077
3	Прочность при растяжении, МПа	KNeighborsRegressor	0.033856	-0.043321

```
# Отрисуем результат

result_plot(y_test_1.values, neigh_1.predict(x_test_1), 'Модуль упругости при растяжении, ГПа', 'KNeighborsRegressor')
result_plot(y_test_2.values, neigh_2.predict(x_test_2), 'Прочность при растяжении, МПа', 'KNeighborsRegressor')
```

Случайный лес, регрессия

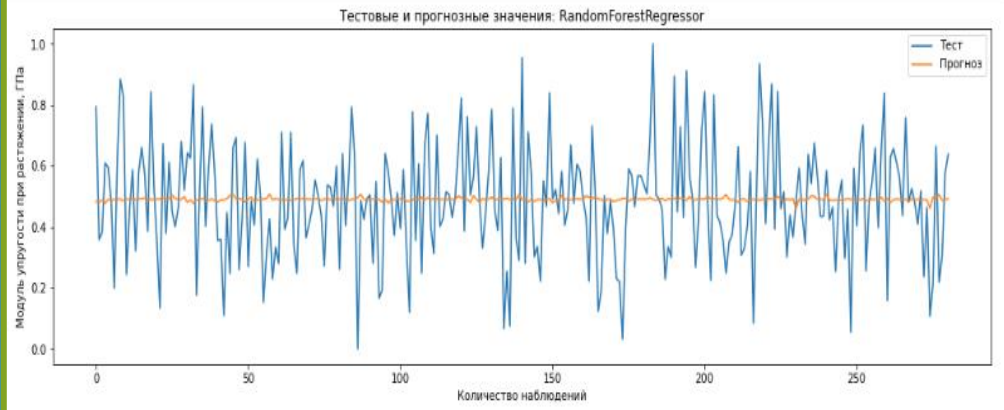
```
# Случайный лес, регрессия.
```

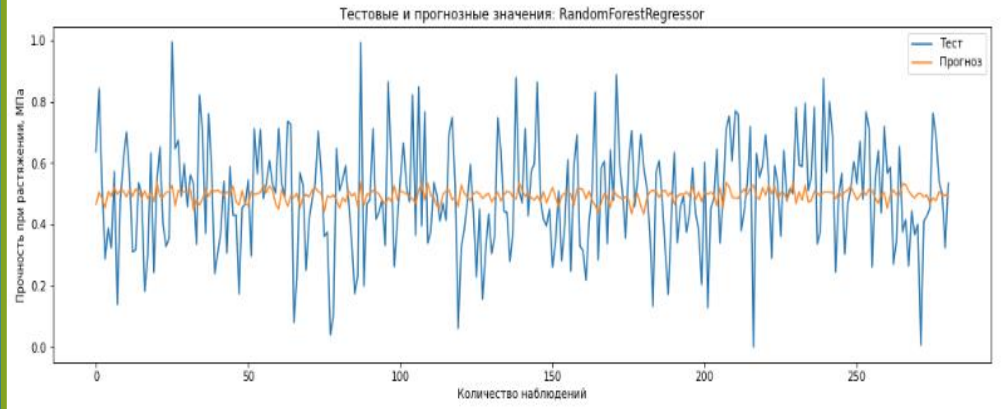
```
%%time

regr_rf = RandomForestRegressor(random_state=14)
regr_rf_params = {
    'n_estimators' : range(1, 100, 5),
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : range(1, 5, 1),
    'criterion':['mse']
}
GSCV_regr_rf_1 = GridSearchCV(regr_rf, regr_rf_params, cv=5, verbose=2)
GSCV_regr_rf_1.fit(x_train_1, y_train_1)
# GSCV_regr_rf_1.best_params_
regr_rf_1 = GSCV_regr_rf_1.best_estimator_

MSE_1 = mean_squared_error(y_test_1, regr_rf_1.predict(x_test_1))
R2_1 = r2_score(y_test_1, regr_rf_1.predict(x_test_1))

GSCV_regr_rf_2 = GridSearchCV(regr_rf, regr_rf_params, cv=5, verbose=2)
GSCV_regr_rf_2.fit(x_train_2, y_train_2)
# GSCV_regr_rf_2.best_params_
regr_rf_2 = GSCV_regr_rf_2.best_estimator_

MSE_2 = mean_squared_error(y_test_2, regr_rf_2.predict(x_test_2))
R2_2 = r2_score(y_test_2, regr_rf_2.predict(x_test_2))

rf_errors = pd.DataFrame({'model_name':'RandomForestRegressor',\
                          'target_var':['Модуль упругости при растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                          'MSE':[MSE_1, MSE_2],\
                          'R2':[R2_1, R2_2]})
df_errors = pd.concat([df_errors, rf_errors], ignore_index=True)
df_errors
```

```
[CV] END criterion=mse, max_depth=4, max_features=log2, n_estimators=91; total time=   0.1s
[CV] END criterion=mse, max_depth=4, max_features=log2, n_estimators=91; total time=   0.1s
[CV] END criterion=mse, max_depth=4, max_features=log2, n_estimators=96; total time=   0.1s
[CV] END criterion=mse, max_depth=4, max_features=log2, n_estimators=96; total time=   0.1s
[CV] END criterion=mse, max_depth=4, max_features=log2, n_estimators=96; total time=   0.1s
[CV] END criterion=mse, max_depth=4, max_features=log2, n_estimators=96; total time=   0.1s
[CV] END criterion=mse, max_depth=4, max_features=log2, n_estimators=96; total time=   0.1s
Wall time: 4min 23s
```

	target_var	model_name	MSE	R2
0	Модуль упругости при растяжении, ГПа	LinearRegression	0.035136	0.014854
1	Прочность при растяжении, МПа	LinearRegression	0.033425	-0.030041
2	Модуль упругости при растяжении, ГПа	KNeighborsRegressor	0.035057	0.017077
3	Прочность при растяжении, МПа	KNeighborsRegressor	0.033856	-0.043321
4	Модуль упругости при растяжении, ГПа	RandomForestRegressor	0.035708	-0.001175
5	Прочность при растяжении, МПа	RandomForestRegressor	0.033061	-0.018838

```
# Отрисуем результат

result_plot(y_test_1.values, regr_rf_1.predict(x_test_1), 'Модуль упругости при растяжении, ГПа', 'RandomForestRegressor')
result_plot(y_test_2.values, regr_rf_2.predict(x_test_2), 'Прочность при растяжении, МПа', 'RandomForestRegressor')
```





Многослойный перцептрон

```
# Многослойный перцептрон
```

```
%%time

neuronet = MLPRegressor(random_state=14)
neuronet_params = {
    'hidden_layer_sizes' : [(100, 100, 50, 25, 12), (144, 144, 72, 36, 12, 1), (12, 12, 12, 12, 12),
                            (144, 144, 144, 72, 72, 36, 36), ()],
    'activation' : ['identity', 'logistic', 'tanh', 'relu'],
    'solver' : ['sgd', 'adam'],
    'max_iter' : [100],
    'learning_rate' : ['constant', 'adaptive', 'invscaling']
}

GSCV_neuronet_1 = GridSearchCV(neuronet, neuronet_params, n_jobs=-1, cv=10)
GSCV_neuronet_1.fit(x_train_1, y_train_1)
# GSCV_neuronet_1.best_params_
neuronet_1 = GSCV_neuronet_1.best_estimator_

MSE_1 = mean_squared_error(y_test_1, neuronet_1.predict(x_test_1))
R2_1 = r2_score(y_test_1, neuronet_1.predict(x_test_1))

GSCV_neuronet_2 = GridSearchCV(neuronet, neuronet_params, n_jobs=-1, cv=10)
GSCV_neuronet_2.fit(x_train_2, y_train_2)
# GSCV_neuronet_2.best_params_
neuronet_2 = GSCV_neuronet_2.best_estimator_

MSE_2 = mean_squared_error(y_test_2, neuronet_2.predict(x_test_2))
R2_2 = r2_score(y_test_2, neuronet_2.predict(x_test_2))

neuronet_errors = pd.DataFrame({'model_name':'MLPRegressor',\
                                'target_var':['Модуль упругости при растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                               'MSE':[MSE_1, MSE_2],\
                               'R2':[R2_1, R2_2]})
df_errors = pd.concat([df_errors, neuronet_errors], ignore_index=True)
df_errors
```

```
Wall time: 18min 30s
```

	target_var	model_name	MSE	R2
0	Модуль упругости при растяжении, ГПа	LinearRegression	0.035136	0.014854
1	Прочность при растяжении, МПа	LinearRegression	0.033425	-0.030041
2	Модуль упругости при растяжении, ГПа	KNeighborsRegressor	0.035057	0.017077
3	Прочность при растяжении, МПа	KNeighborsRegressor	0.033856	-0.043321
4	Модуль упругости при растяжении, ГПа	RandomForestRegressor	0.035708	-0.001175
5	Прочность при растяжении, МПа	RandomForestRegressor	0.033061	-0.018838
6	Модуль упругости при растяжении, ГПа	MLPRegressor	0.035717	-0.001440
7	Прочность при растяжении, МПа	MLPRegressor	0.034627	-0.067105

Прогнозные значения различаются только в пятом знаке после запятой. Результат плохой.

```
# Отрисуем результат

result_plot(y_test_1.values, neuronet_1.predict(x_test_1), 'Модуль упругости при растяжении, ГПа', 'MLPRegressor')
result_plot(y_test_2.values, neuronet_2.predict(x_test_2), 'Прочность при растяжении, МПа', 'MLPRegressor')
```

Лассо регрессия

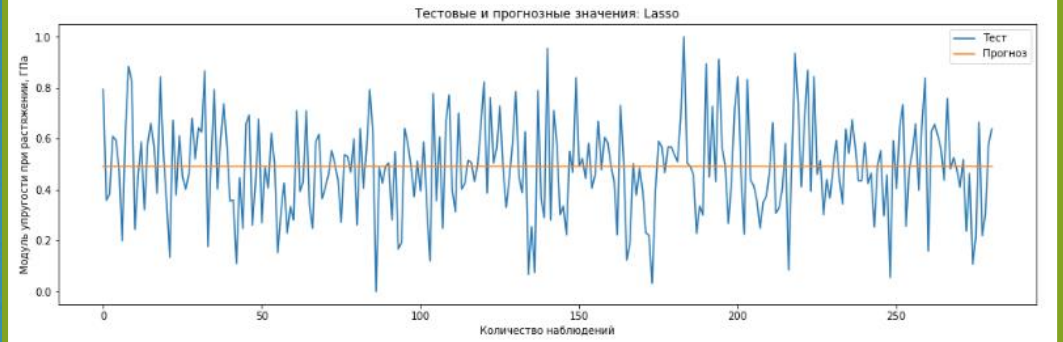
```
# Лассо регрессия
```

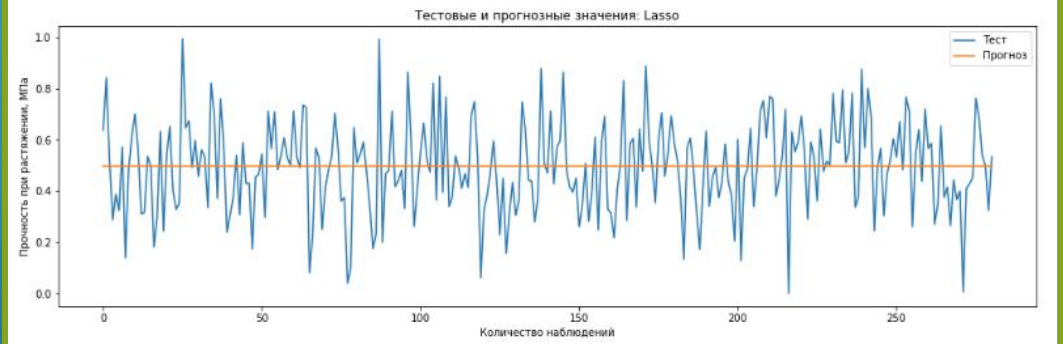
```
lasso = Lasso(random_state=14)
lasso_params = {
   'alpha': np.linspace(0, 1, 100)
}
GSCV_lasso_1 = GridSearchCV(lasso, lasso_params, cv=10, verbose=2)
GSCV_lasso_1.fit(x_train_1, y_train_1)
# GSCV_lasso_1.best_params_

lasso_1 = GSCV_lasso_1.best_estimator_

MSE_1 = mean_squared_error(y_test_1, lasso_1.predict(x_test_1))
R2_1 = r2_score(y_test_1, lasso_1.predict(x_test_1))

GSCV_lasso_2 = GridSearchCV(lasso, lasso_params, cv=10, verbose=2)
GSCV_lasso_2.fit(x_train_2, y_train_2)
# GSCV_lasso_2.best_params_

lasso_2 = GSCV_lasso_2.best_estimator_
MSE_2 = mean_squared_error(y_test_2, lasso_2.predict(x_test_2))
R2_2 = r2_score(y_test_2, lasso_2.predict(x_test_2))

lasso_errors = pd.DataFrame({'model_name':'Lasso',\
                             'target_var':['Модуль упругости при растяжении, ГПа', 'Прочность при растяжении, МПа'],\
                            'MSE':[MSE_1, MSE_2],\
                            'R2':[R2_1, R2_2]})
df_errors = pd.concat([df_errors, lasso_errors], ignore_index=True)
df_errors
```

```
[CV] END .....................................alpha=1.0; total time=   0.0s
[CV] END .....................................alpha=1.0; total time=   0.0s
```

	target_var	model_name	MSE	R2
0	Модуль упругости при растяжении, ГПа	LinearRegression	0.035136	0.014854
1	Прочность при растяжении, МПа	LinearRegression	0.033425	-0.030041
2	Модуль упругости при растяжении, ГПа	KNeighborsRegressor	0.035057	0.017077
3	Прочность при растяжении, МПа	KNeighborsRegressor	0.033856	-0.043321
4	Модуль упругости при растяжении, ГПа	RandomForestRegressor	0.035708	-0.001175
5	Прочность при растяжении, МПа	RandomForestRegressor	0.033061	-0.018838
6	Модуль упругости при растяжении, ГПа	MLPRegressor	0.035717	-0.001440
7	Прочность при растяжении, МПа	MLPRegressor	0.034627	-0.067105
8	Модуль упругости при растяжении, ГПа	Lasso	0.035697	-0.000872
9	Прочность при растяжении, МПа	Lasso	0.032464	-0.000439

```
# Отрисуем результат

result_plot(y_test_1.values, lasso_1.predict(x_test_1), 'Модуль упругости при растяжении, ГПа', 'Lasso')
result_plot(y_test_2.values, lasso_2.predict(x_test_2), 'Прочность при растяжении, МПа', 'Lasso')
```





Результаты обучения

- Итоговый датасет ошибок:

	target_var	model_name	MSE	R2
0	Модуль упругости при растяжении, ГПа	LinearRegression	0.035136	0.014854
1	Прочность при растяжении, МПа	LinearRegression	0.033425	-0.030041
2	Модуль упругости при растяжении, ГПа	KNeighborsRegressor	0.035057	0.017077
3	Прочность при растяжении, МПа	KNeighborsRegressor	0.033856	-0.043321
4	Модуль упругости при растяжении, ГПа	RandomForestRegressor	0.035708	-0.001175
5	Прочность при растяжении, МПа	RandomForestRegressor	0.033061	-0.018838
6	Модуль упругости при растяжении, ГПа	MLPRegressor	0.035717	-0.001440
7	Прочность при растяжении, МПа	MLPRegressor	0.034627	-0.067105
8	Модуль упругости при растяжении, ГПа	Lasso	0.035697	-0.000872
9	Прочность при растяжении, МПа	Lasso	0.032464	-0.000439

Написание нейронной сети на Tensorflow.Keras для целевой переменной «соотношение матрица-наполнитель»

Предобработка данных и задание архитектуры:

```
# Определим входы и выход для модели

target_var = df['Соотношение матрица-наполнитель']
train_vars = df.loc[:, df.columns != 'Соотношение матрица-наполнитель']

# Разбиваем выборки на обучающую и тестовую
x_train, x_test, y_train, y_test = train_test_split(train_vars, target_var, test_size=0.3, random_state=14)
```

```
# нормализация

x_train_normalizer = tf.keras.layers.Normalization(axis=-1)
x_train_normalizer.adapt(np.array(x_train))
```

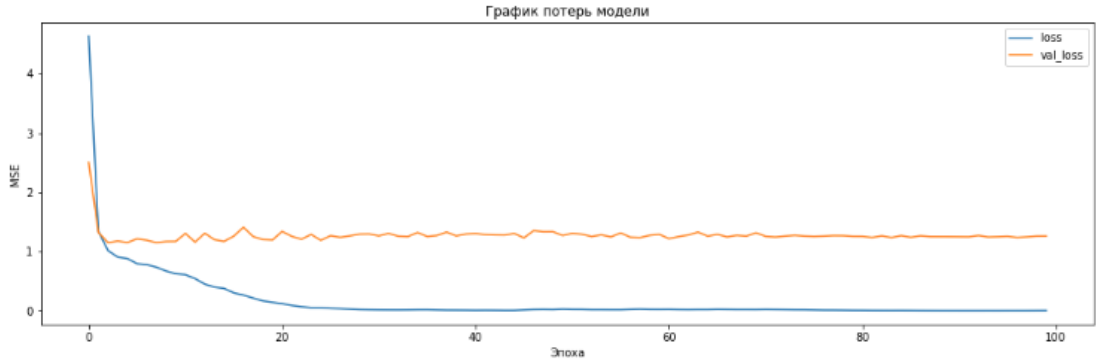
```
# Сконфигурируем модель, зададим слои

model = tf.keras.Sequential([x_train_normalizer, layers.Dense(128, activation='relu'),
                                                 layers.Dense(128, activation='relu'),
                                                 layers.Dense(128, activation='relu'),
                                                 layers.Dense(64, activation='relu'),
                                                 layers.Dense(32, activation='relu'),
                                                 layers.Dense(16, activation='relu'),
                                                 layers.Dense(1)
                                                ])

model.compile(optimizer=tf.keras.optimizers.Adam(0.001), loss='mean_squared_error')
```

```
# Архитектура модели

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 12)               25
 n)

 dense (Dense)               (None, 128)               1664

 dense_1 (Dense)             (None, 128)               16512

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 32)                2080

 dense_6 (Dense)             (None, 16)                528

 dense_7 (Dense)             (None, 1)                 17

=================================================================
Total params: 62,106
Trainable params: 62,081
Non-trainable params: 25
_________________________________________________________________
```

Результаты работы модели

```python
# Отрисовка потерь на тренировочной и тестовой выборках
def model_losses(model_hist):
    plt.figure(figsize=(17,5))
    plt.plot(model_hist.history['loss'])
    plt.plot(model_hist.history['val_loss'])
    plt.title('График потерь модели')
    plt.ylabel('MSE')
    plt.xlabel('Эпоха')
    plt.legend(['loss', 'val_loss'], loc='best')
    plt.show()
model_losses(model_hist)
```




```python
# Отрисуем результат работы модели

result_plot(y_test.values, model.predict(x_test.values), 'Соотношение матрица/наполнитель', 'Keras_neuronet')
```


```python
# оценка модели MSE
model.evaluate(x_test, y_test, verbose=1)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 1.2517

1.251726746559143
```

```python
# Обучим модель

model_hist = model.fit(x_train, y_train, epochs=100, verbose=1, validation_split = 0.2)
```

```
17/17 [==============================] - 0s 5ms/step - loss: 7.2893e-04 - val_loss: 1.2483
Epoch 92/100
17/17 [==============================] - 0s 5ms/step - loss: 7.3510e-04 - val_loss: 1.2471
Epoch 93/100
17/17 [==============================] - 0s 5ms/step - loss: 6.6358e-04 - val_loss: 1.2690
Epoch 94/100
17/17 [==============================] - 0s 6ms/step - loss: 7.7644e-04 - val_loss: 1.2440
Epoch 95/100
17/17 [==============================] - 0s 5ms/step - loss: 6.8000e-04 - val_loss: 1.2488
Epoch 96/100
17/17 [==============================] - 0s 5ms/step - loss: 9.5300e-04 - val_loss: 1.2552
Epoch 97/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0010 - val_loss: 1.2364
Epoch 98/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0011 - val_loss: 1.2462
Epoch 99/100
17/17 [==============================] - 0s 5ms/step - loss: 0.0017 - val_loss: 1.2588
Epoch 100/100
17/17 [==============================] - 0s 6ms/step - loss: 0.0023 - val_loss: 1.2598
```

Ошибки модели MSE и R^2:

10	Соотношение матрица/наполнитель	Keras_neuronet	1.251727	-0.488577

```python
# Сохраним модель для разработки веб-приложения для
# прогнозирования соотношения "матрица-наполнитель" в фреймворке Flask

model.save('flaskProject/saved_models/keras_model')
```

```
INFO:tensorflow:Assets written to: flaskProject/saved_models/keras_model\assets
```

Разработка web-приложения

Visual Studio

Heroku

Спасибо за внимание!