ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА по курсу «DATA SCIENCE»

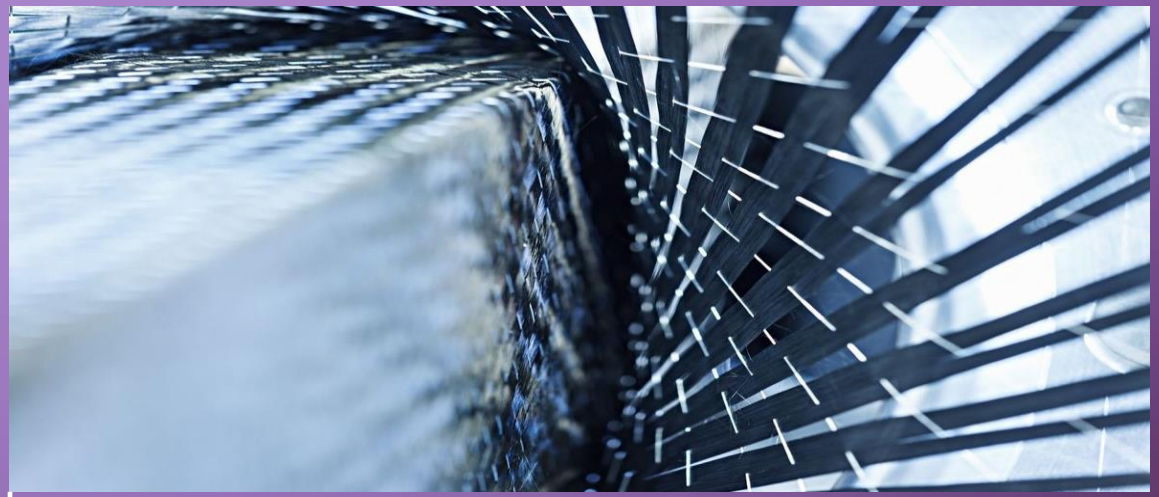
Слушатель: Кудрявцева Мария Валерьевна

ПОСТАНОВКА ЗАДАЧИ:

Цель выпускной квалификационной работы – я изучение способов прогнозирования конечных свойств новых композиционных материалов и разработка моделей для выполнения прогнозов.

Для достижения данной цели необходимо решение следующих **задач**:

- разработка алгоритма машинного обучения для прогноза значений модуля упругости при растяжении и прочности при растяжении;
- создание нейронной сеть для рекомендации соотношения «матрица-наполнитель».



Справочно:

Композиционный материал представляет собой неоднородный сплошной материал, состоящий из двух или более компонентов, среди которых можно выделить армирующие элементы (наполнители), обеспечивающие необходимые механические характеристики материала, и матрицу (или связующее), обеспечивающую совместную работу армирующих элементов. Матрица может быть металлическая, керамическая, углеродная, полимерная и т.д. Наполнитель может состоять из частиц, волокон, тканей или листов.

ИСХОДНЫЕ ДАННЫЕ ДЛЯ АНАЛИЗА

- Использованы производственные данные Центра НТИ «Цифровое материаловедение: новые материалы и вещества».
- Информация представлена в виде двух файлов формата excel: X_bp.xlsx (характеристики базальтопластика) и X_nup.xlsx (характеристики нашивки из углепластика).
- Для работы указанные датасеты объединены в один по индексу по типу объединения INNER с удалением 17 строк второго датасета.

In [11]:
```
# Объединеним датесеты в один
# Объединение выполним с помощью функции merge() по индексу с типом объединения INNER
dataset = pd.merge(dataset_bp, dataset_nup,
                   left_index=True,
                   right_index=True,
                   how = "inner")


# Выведем 5 первых позиций нового датасета
dataset.head()
```

Out[11]:

	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	наш
0	1.857143	2030.0	738.736842	30.00	22.267857	100.000000	210.0	70.0	3000.0	220.0	0	
1	1.857143	2030.0	738.736842	50.00	23.750000	284.615385	210.0	70.0	3000.0	220.0	0	
2	1.857143	2030.0	738.736842	49.90	33.000000	284.615385	210.0	70.0	3000.0	220.0	0	
3	1.857143	2030.0	738.736842	129.00	21.250000	300.000000	210.0	70.0	3000.0	220.0	0	
4	2.771331	2030.0	753.000000	111.86	22.267857	284.615385	210.0	70.0	3000.0	220.0	0	

In [12]:
```
# Посмотрим размерность объединенного датасета
dataset.shape
```

Out[12]: (1023, 13)

РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ

- Информация обо всех столбцах датасета с указанием типов данных
- Анализ числовых статистик (количество элементов, среднее арифметическое, медиана, стандартное отклонение, минимальное и максимальное значения, перцентили)
- Проверка наличия пропусков в датасете
- Подсчет уникальных значений для каждой характеристики

In [19]:
```
# Посмотрим информацию обо всех столбцах датасета
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1023 entries, 0 to 1022
Data columns (total 13 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Соотношение матрица-наполнитель    1023 non-null   float64
 1   Плотность, кг/м3                   1023 non-null   float64
 2   модуль упругости, ГПа              1023 non-null   float64
 3   Количество отвердителя, м.%        1023 non-null   float64
 4   Содержание эпоксидных групп,%_2    1023 non-null   float64
 5   Температура вспышки, С_2           1023 non-null   float64
 6   Поверхностная плотность, г/м2      1023 non-null   float64
 7   Модуль упругости при растяжении, ГПа  1023 non-null   float64
 8   Прочность при растяжении, МПа      1023 non-null   float64
 9   Потребление смолы, г/м2            1023 non-null   float64
 10  Угол нашивки, град                 1023 non-null   int64  
 11  Шаг нашивки                        1023 non-null   float64
 12  Плотность нашивки                  1023 non-null   float64
dtypes: float64(12), int64(1)
memory usage: 111.9 KB
```

Датасет состоит из 13 столбцов.

Пропусков в датасете нет, каждый столбец содержит 1023 значения.

Тип данных большинства столбцов - float64 (числа с плавающей точкой), только столбец "Угол нашивки, град" имеет тип int64 (целые числа).

In [26]:
```
# Посмотрим числовые статистики с помощью метода describe()
df.describe()
```

Out[26]:

	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	У наши
count	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000000	1023.000
mean	2.930366	1975.734888	739.923233	110.570769	22.244390	285.882151	482.731833	73.328571	2466.922843	218.423144	0.491
std	0.913222	73.729231	330.231581	28.295911	2.406301	40.943260	281.314690	3.118983	485.628006	59.735931	0.500
min	0.389403	1731.764635	2.436909	17.740275	14.254985	100.000000	0.603740	64.054061	1036.856605	33.803026	0.000
25%	2.317887	1924.155467	500.047452	92.443497	20.608034	259.066528	266.816645	71.245018	2135.850448	179.627520	0.000
50%	2.906878	1977.621657	739.664328	110.564840	22.230744	285.896812	451.864365	73.268805	2459.524526	219.198882	0.000
75%	3.552660	2021.374375	961.812526	129.730366	23.961934	313.002106	693.225017	75.356612	2767.193119	257.481724	1.000
max	5.591742	2207.773481	1911.536477	198.953207	33.000000	413.273418	1399.542362	82.682051	3848.436732	414.590628	1.000

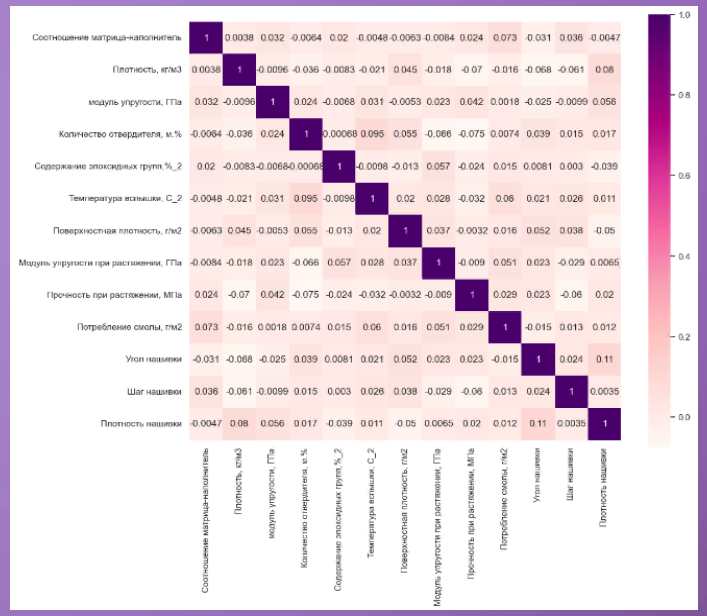
In [21]:
```
# Посмотрим количество уникальных значений по всем столбцам датасета с помощью функции nunique()
df.nunique()
```

Out[21]:
```
Соотношение матрица-наполнитель          1014
Плотность, кг/м3                         1013
модуль упругости, ГПа                    1020
Количество отвердителя, м.%              1005
Содержание эпоксидных групп,%_2          1004
Температура вспышки, С_2                 1003
Поверхностная плотность, г/м2            1004
Модуль упругости при растяжении, ГПа     1004
Прочность при растяжении, МПа            1004
Потребление смолы, г/м2                  1003
Угол нашивки, град                          2
Шаг нашивки                               989
Плотность нашивки                         988
dtype: int64
```

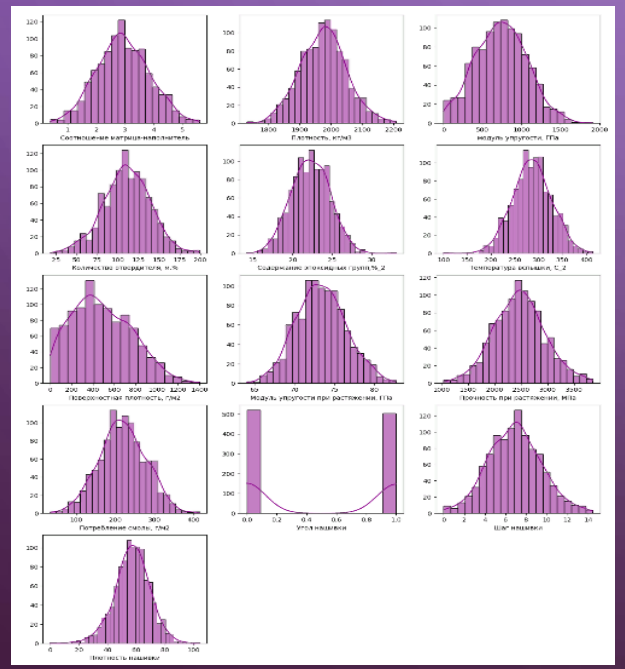
Большинство столбцов содержит около тысячи уникальных значений, кроме столбца "Угол нашивки, град", где уникальных значений всего 2.

ВИЗУАЛИЗАЦИЯ ИСХОДНЫХ ДАННЫХ

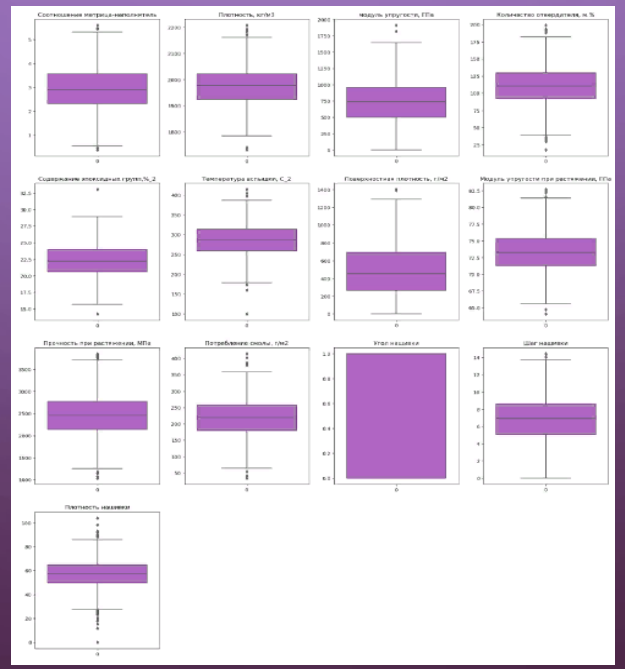
Тепловая карта



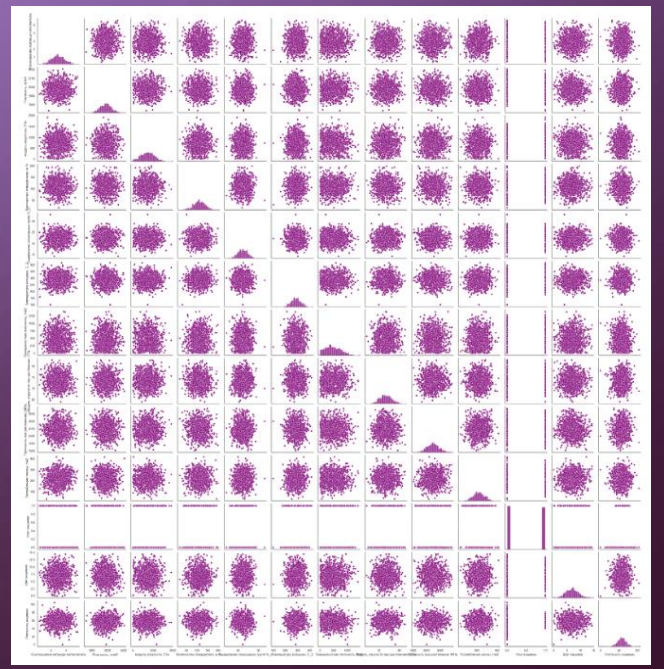
Гистограммы и графики плотности



Диаграммы «ящик с усами»



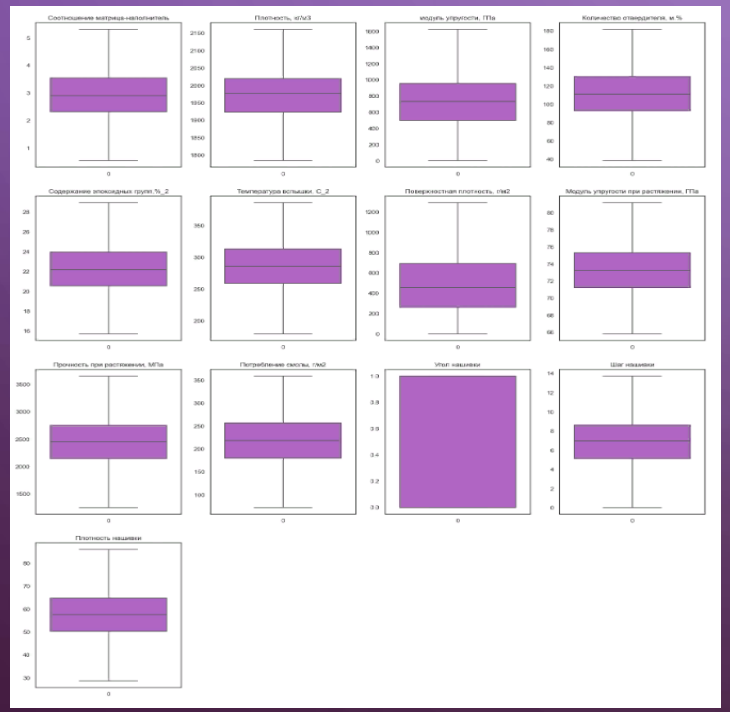
Попарные графики рассеяния точек



ПРЕДОБРАБОТКА ДАННЫХ

Поиск и удаление выбросов

1) Стандартное отклонение (правило трех сигм)
2) Метод межквартильного диапазона (IQR)



Нормализация и стандартизация данных

Тест Шапиро-Уилка на нормальность

```
In [50]: # Проведем тест Шапиро-Уилка на нормальность
for col in df_clean.columns:
    print(df_clean[col].name, shapiro(df_clean[col]))
```

```
Соотношение матрица-наполнитель ShapiroResult(statistic=0.9971880316734314, pvalue=0.10904344916343689)
Плотность, кг/м3 ShapiroResult(statistic=0.997011661529541, pvalue=0.08352091163396835)
модуль упругости, ГПа ShapiroResult(statistic=0.9952543497392273, pvalue=0.005813030526041946)
Количество отвердителя, м.% ShapiroResult(statistic=0.9966756105422974, pvalue=0.05000615119934082)
Содержание эпоксидных групп,%_2 ShapiroResult(statistic=0.9977133870124817, pvalue=0.23541033267974854)
Температура вспышки, С_2 ShapiroResult(statistic=0.9971266984939575, pvalue=0.09941922873258591)
Поверхностная плотность, г/м2 ShapiroResult(statistic=0.9776217937469482, pvalue=1.0688074730813568e-10)
Модуль упругости при растяжении, ГПа ShapiroResult(statistic=0.9955782890319824, pvalue=0.009416559711098671)
Прочность при растяжении, МПа ShapiroResult(statistic=0.9973730444908142, pvalue=0.14375117421150208)
Потребление смолы, г/м2 ShapiroResult(statistic=0.9955164790153503, pvalue=0.008584197610616684)
Угол нашивки ShapiroResult(statistic=0.6364129781723022, pvalue=2.8589291269154918e-40)
Шаг нашивки ShapiroResult(statistic=0.9980173110961914, pvalue=0.3559962809085846)
Плотность нашивки ShapiroResult(statistic=0.9967383742332458, pvalue=0.05505112186074257)
```

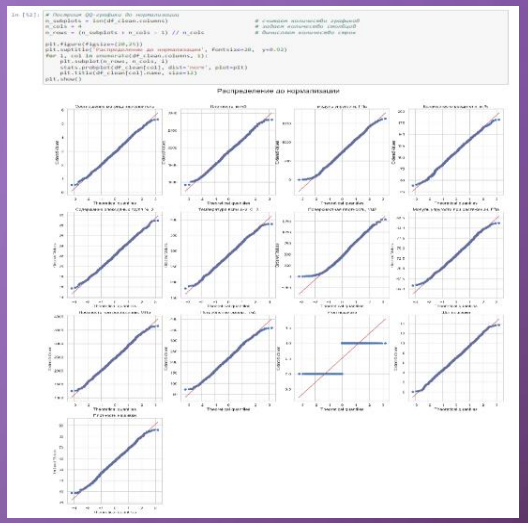
В данном случае нулевая гипотеза отвергается (p-value < 0.05) в случаях:

- модуль упругости, ГПа
- Поверхностная плотность, г/м2
- Модуль упругости при растяжении, ГПа
- Потребление смолы, г/м2
- Угол нашивки

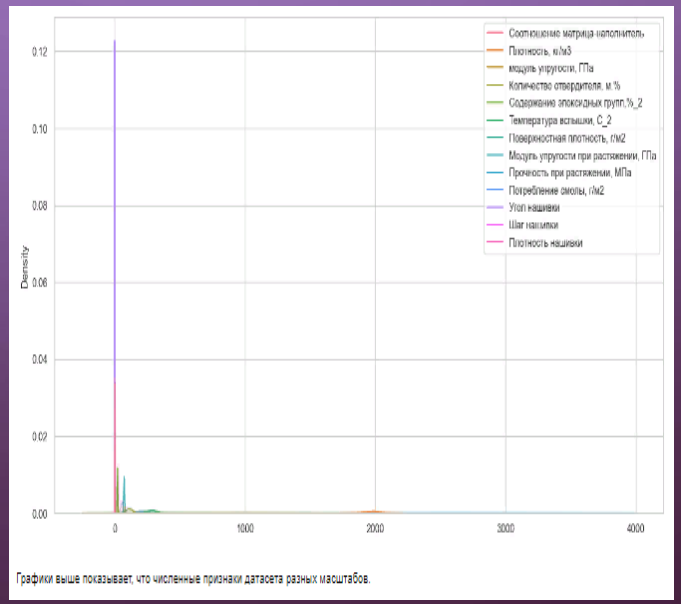
что подразумевает, что распределения НЕ является нормальными.

Однако, проверка нормальности статистическими тестами является очень строгой, т.к. идет сравнение с идеальным распределением. Поэтому несмотря на то, что статистический тест говорит о ненормальности распределения, стоит посмотреть на гистограмму распределения.

QQ-графики

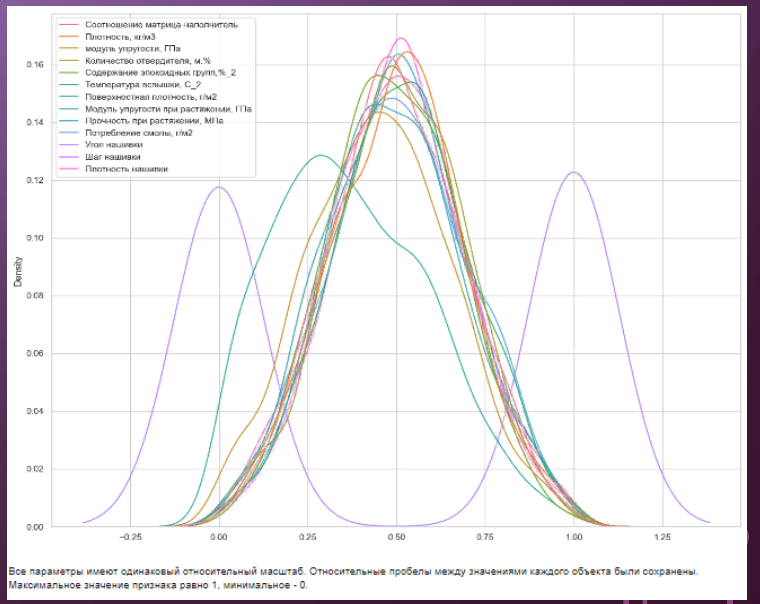


Распределение переменных датасета



Графики выше показывает, что численные признаки датасета разных масштабов.

Распределение переменных датасета после масштабирования MinMaxScaler



Все параметры имеют одинаковый относительный масштаб. Относительные пробелы между значениями каждого объекта были сохранены. Максимальное значение признака равно 1, минимальное - 0.

РАЗРАБОТКА И ОБУЧЕНИЕ МОДЕЛЕЙ

Прогнозирование модуля упругости при растяжении

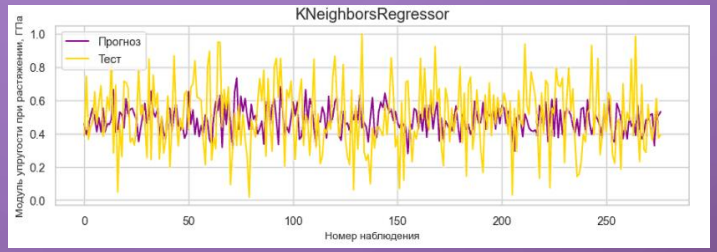
- Разделение данных на обучающую и тестовую выборку

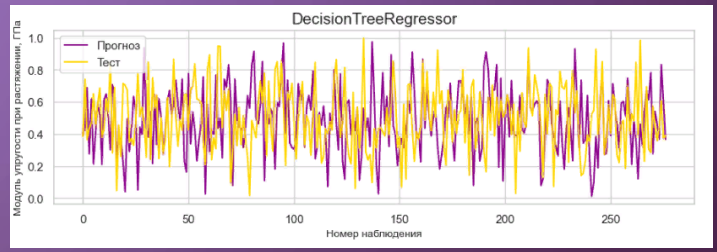
```
# Разделим датасет на тренировочную и тестовую выборки.
# При построении модели 30% данных оставим на тестирование модели, на остальных происходит обучение моделей.
X_train_elastic, X_test_elastic, y_train_elastic, y_test_elastic = train_test_split(
                                                                   df_norm.drop(['Модуль упругости при растяжении, ГПа', 'Прочнос
                                                                   df_norm[['Модуль упругости при растяжении, ГПа']],
                                                                   test_size = 0.3,
                                                                   random_state = 42,
                                                                   shuffle = True
                                                                   )
```

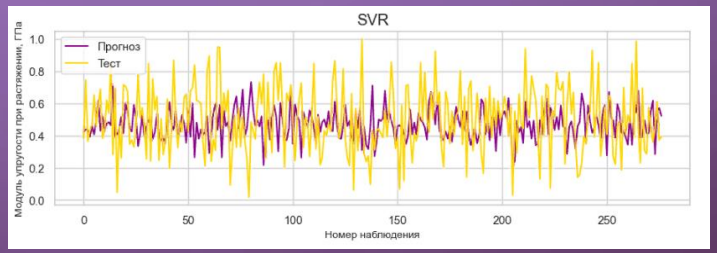
```
# Посмотрим размерность тренировочной и тестовой выборок
print('Размер тренировочного датасета: {}\nРазмер тестового датасета:{}'.format(X_train_elastic.shape, X_test_elastic.shape))
```

```
Размер тренировочного датасета: (645, 11)
Размер тестового датасета:(277, 11)
```

- Анализ работы различных моделей на стандартных параметрах
  1. Метод К-ближайших соседей
  2. Метод опорных векторов
  3. Линейная регрессия
  4. Дерево решений
  5. AdaBoost
  6. Градиентный бустинг
  7. XGBoost
  8. Случайный лес
  9. Стохастический градиентный спуск
  10. Метод регрессии «Lasso»







ПОИСК ГИПЕРПАРАМЕТРОВ МОДЕЛЕЙ

Метод поиска по сетке GridSearchCV() с перекрестной проверкой, количество блоков равно 10

```
# Создаем словарь с наборами гиперпараметров всех моделей
all_params = {'kneighborsregressor':       {'kneighborsregressor__n_neighbors': [i for i in range(1, 201, 2)],
                                            'kneighborsregressor__weights':     ['uniform', 'distance'],
                                            'kneighborsregressor__algorithm':   ['auto', 'ball_tree', 'kd_tree', 'brute']
                                           },
              'svr':                       {'svr__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
                                            'svr__C':      [0.01, 0.1, 1],
                                            'svr__gamma':  [0.01, 0.1, 1]
                                           },
              'linearregression':          {'linearregression__fit_intercept': [True, False]},
              'decisiontreeregressor':     {'decisiontreeregressor__max_depth':        [3, 5, 7, 9, 11, 13, 15],
                                            'decisiontreeregressor__min_samples_leaf': [1, 2, 5, 10, 20, 50, 100, 150, 200],
                                            'decisiontreeregressor__min_samples_split': [200, 250, 300],
                                            'decisiontreeregressor__max_features':     ['auto', 'sqrt', 'log2']
                                           },
              'adaboostregressor':         {'adaboostregressor__base_estimator__max_depth':       [i for i in range(2, 11, 2)],
                                            'adaboostregressor__base_estimator__min_samples_leaf': [5,  10],
                                            'adaboostregressor__n_estimators':                     [10, 50, 100, 250, 1000],
                                            'adaboostregressor__learning_rate':                    [0.01, 0.05, 0.1, 0.5]
                                           },
              'gradientboostingregressor': {'gradientboostingregressor__learning_rate': [0.01,  0.02, 0.03, 0.04],
                                            'gradientboostingregressor__subsample'    : [0.9, 0.5, 0.2, 0.1],
                                            'gradientboostingregressor__n_estimators' : [100, 500, 1000, 1500],
                                            'gradientboostingregressor__max_depth'    : [4, 6, 8, 10]
                                           },
              'xgbregressor':              {'xgbregressor__learning_rate':    [0.05, 0.10, 0.15],
                                            'xgbregressor__max_depth':        [ 3, 4, 5, 6, 8],
                                            'xgbregressor__min_child_weight': [ 1, 3, 5, 7],
                                            'xgbregressor__gamma':            [ 0.0, 0.1, 0.2],
                                            'xgbregressor__colsample_bytree': [ 0.3, 0.4]
                                           },
              'randomforestregressor':     {'randomforestregressor__n_estimators':     [30, 100, 200, 300, 500],
                                            'randomforestregressor__max_depth':        [1, 2, 3, 4, 5, 6, 7, 8],
                                            'randomforestregressor__min_samples_leaf': [1, 2],
                                            'randomforestregressor__max_features':     ['auto', 'sqrt', 'log2']
                                           },
              'sgdregressor':              {'sgdregressor__penalty': ['l2', 'l1', 'elasticnet', None],
                                            'sgdregressor__alpha':   [0.0001, 0.001, 0.01, 0.1]
                                           },
              'lasso':                     {'lasso__alpha': [0.01, 0.02, 0.1, 0.2, 0.03, 0.3, 0.05, 0.5, 0.07, 0.7, 1]}
             }
```

```
    # Выполняем обучение и оценку качества с помощью кросс-валидации,
    model = GridSearchCV(estimator = pipe, param_grid = current_params,
                         scoring = 'r2', cv = 10, n_jobs = -1)
    model.fit(x_train, y_train)

    print('{} Лучшее значение R2 на тренировочной выборке: {:.4f}'.format(all_models[regressor], model.score(x_train, y_train)))
    print('{} Лучшее значение R2 на тестовой выборке {:.4f}'.format(all_models[regressor], model.score(x_test, y_test)))
    print('{} Лучшее значение R2 на перекрестной проверке: {:.4f}'.format(all_models[regressor], model.best_score_))
    print('{} Лучшие параметры модели: {}'.format(all_models[regressor], model.best_params_))
    print('\n_____________________________________________________________________________\n')
    if model.score(x_test, y_test) > best_score:
            best_score = model.score(x_test, y_test)
            best_model = model.best_estimator_
            best_regressor = regressor
print('Регрессор с лучшим значением R2 = {:.4f} на тестовой выборке: {}\n'.format(best_score, all_models[best_regressor]))
print('Лучший алгоритм:\n{}\n'.format(best_model))
```

```
In [75]: Model_Selection(X_train_elastic, np.ravel(y_train_elastic), X_test_elastic, np.ravel(y_test_elastic))
```

```
KNeighborsRegressor() Лучшее значение R2 на тренировочной выборке: 0.0091
KNeighborsRegressor() Лучшее значение R2 на тестовой выборке -0.0105
KNeighborsRegressor() Лучшее значение R2 на перекрестной проверке: -0.0021
KNeighborsRegressor() Лучшие параметры модели: {'kneighborsregressor__algorithm': 'auto', 'kneighborsregressor__n_neighbors':
199, 'kneighborsregressor__weights': 'uniform'}

_____________________________________________________________________________

SVR() Лучшее значение R2 на тренировочной выборке: 0.0326
SVR() Лучшее значение R2 на тестовой выборке -0.0146
SVR() Лучшее значение R2 на перекрестной проверке: -0.0031
SVR() Лучшие параметры модели: {'svr__C': 0.01, 'svr__gamma': 1, 'svr__kernel': 'rbf'}

_____________________________________________________________________________

LinearRegression() Лучшее значение R2 на тренировочной выборке: 0.0172
LinearRegression() Лучшее значение R2 на тестовой выборке -0.0245
LinearRegression() Лучшее значение R2 на перекрестной проверке: -0.0307
LinearRegression() Лучшие параметры модели: {'linearregression__fit_intercept': True}
```

```
Регрессор с лучшим значением R2 = -0.0009 на тестовой выборке: AdaBoostRegressor(base_estimator=DecisionTreeRegressor())

Лучший алгоритм:
Pipeline(steps=[('adaboostregressor',
                 AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=2,
                                                                        min_samples_leaf=10),
                                   learning_rate=0.01, n_estimators=10))])
```

```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor()) Лучшее значение R2 на тренировочной выборке: 0.0128
AdaBoostRegressor(base_estimator=DecisionTreeRegressor()) Лучшее значение R2 на тестовой выборке -0.0009
AdaBoostRegressor(base_estimator=DecisionTreeRegressor()) Лучшее значение R2 на перекрестной проверке: -0.0168
AdaBoostRegressor(base_estimator=DecisionTreeRegressor()) Лучшие параметры модели: {'adaboostregressor__base_estimator__max_d
epth': 2, 'adaboostregressor__base_estimator__min_samples_leaf': 10, 'adaboostregressor__learning_rate': 0.01, 'adaboostregre
ssor__n_estimators': 10}
```

ОЦЕНКА КАЧЕСТВА РАБОТЫ МОДЕЛЕЙ

- Коэффициент детерминации (R2) показывает силу связи между двумя случайными величинами. Если модель всегда предсказывает точно, метрика равна 1. Для тривиальной модели – 0. Значение метрики может быть отрицательно, если модель предсказывает хуже, чем тривиальная.
- Средняя абсолютная ошибка (mean absolute error, MAE) показывает среднее абсолютное отклонение предсказанных значений от реальных. Чем выше значение MAE, тем модель хуже. У идеальной модели MAE = 0. MAE очень легко интерпретировать – на сколько в среднем ошибается модель.

In [73]:
```
# Создадим функцию, которая обрабатывает различные модели, переданные в качестве аргумента, и выдает для них метрики эффективнос
# Метрики вычисляются как для тестовых данных, так и для тренировочных данных - тренировочные метрики находятся в ()
def Model_Comparision_Train_Test(AllModels, x_train, y_train, x_test, y_test):
    return_df = pd.DataFrame(columns=['Model', 'R2_score', 'MAE', 'MSE', 'RMSE', 'MAPE'])
    for myModel in AllModels:
        myModel.fit(x_train, y_train)

        # Предсказание и вычисление метрик для тренирвоочного набора данных
        y_pred_train = myModel.predict(x_train)
        r2_score_train, mae_train, mse_train, rmse_train, mape_train = extract_metrics_from_predicted(y_train, y_pred_train)

        # Предсказание и вычисление метрик для тестового набора данных
        y_pred_test = myModel.predict(x_test)
        r2_score_test, mae_test, mse_test, rmse_test, mape_test = extract_metrics_from_predicted(y_test, y_pred_test)

        # Создадим обобщающий датафрейм для всех рассчитанных метрик
        summary = pd.DataFrame([[type(myModel).__name__,
                                 ''.join([str(round(r2_score_test,4)), " (", str(round(r2_score_train,4)), ")"]),
                                 ''.join([str(round(mae_test,2)), " (", str(round(mae_train,2)), ")"]),
                                 ''.join([str(round(mse_test,2)), " (", str(round(mse_train,2)), ")"]),
                                 ''.join([str(round(rmse_test,2)), " (", str(round(rmse_train,2)), ")"]),
                                 ''.join([str(round(mape_test,2)), " (", str(round(mape_train,2)), ")"])]],
                                columns=['Model', 'R2_score', 'MAE', 'MSE', 'RMSE', 'MAPE'])
        return_df = pd.concat([return_df, summary], axis=0)

    # Установим в качестве индекса столбец с названием модели
    return_df.set_index('Model', inplace=True)
    return(return_df)

# Создадим функцию для расчета метрик
def extract_metrics_from_predicted(y_true, y_pred):
    r2 = r2_score(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mape = mean_absolute_percentage_error(y_true, y_pred)
    return (r2, mae, mse, rmse, mape)
```

Метрики для тестовых и тренировочных данных после подбора гиперпараметров – тренировочные метрики находятся в () – для прогнозирования модуля упругости при растяжении

In [108]:
```
# Рассчитаем метрики для моделей с их лучшими параметрами:
KNR_best = KNeighborsRegressor(algorithm='auto', n_neighbors=199, weights='uniform')
SVReg_best = SVR(C=0.01, gamma=1, kernel='rbf')
LR_best = LinearRegression(fit_intercept=True)
DTR_best = DecisionTreeRegressor(max_depth=9, max_features='sqrt', min_samples_leaf=200, min_samples_split=250)
Abr_best = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=2, min_samples_leaf=10), learning_rate=0.01, n_estima
Gbr_best = GradientBoostingRegressor(learning_rate=0.01, max_depth=4, n_estimators=100, subsample=0.5)
XgbR_best = XGBRegressor(colsample_bytree=0.3, gamma=0.2, learning_rate=0.05, max_depth=3, min_child_weight=5)
RFR_best = RandomForestRegressor(max_depth=1, max_features='log2', min_samples_leaf=2, n_estimators=100)
SGDR_best = SGDRegressor(alpha=0.1, penalty='l1')
LassoR_best = Lasso(alpha=0.01)

Model_Comparision_Train_Test([KNR_best, SVReg_best, LR_best, DTR_best, Abr_best, Gbr_best, XgbR_best, RFR_best, SGDR_best, LassoR
```

Out[108]:

Model	R2_score	MAE	MSE	RMSE	MAPE
KNeighborsRegressor	-0.0105 (0.0091)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.65 (3324567063626.71)
SVR	-0.0146 (0.0326)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.65 (3293971898525.6)
LinearRegression	-0.0245 (0.0172)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.65 (3530183592566.66)
DecisionTreeRegressor	0.006 (0.0061)	0.17 (0.15)	0.04 (0.04)	0.2 (0.19)	0.65 (3479792181714.39)
AdaBoostRegressor	-0.0068 (0.0171)	0.17 (0.15)	0.04 (0.04)	0.2 (0.19)	0.66 (3400849732982.94)
GradientBoostingRegressor	-0.0104 (0.1557)	0.17 (0.14)	0.04 (0.03)	0.21 (0.18)	0.65 (2976673273693.22)
XGBRegressor	-0.0218 (0.0918)	0.17 (0.15)	0.04 (0.03)	0.21 (0.18)	0.66 (3274354675248.97)
RandomForestRegressor	-0.0079 (0.0184)	0.17 (0.15)	0.04 (0.04)	0.21 (0.19)	0.65 (3443052806421.78)
SGDRegressor	-0.01 (-0.0)	0.17 (0.16)	0.04 (0.04)	0.21 (0.19)	0.65 (3354326516915.18)
Lasso	-0.0089 (0.0)	0.17 (0.16)	0.04 (0.04)	0.21 (0.19)	0.65 (3362407323544.2)

НЕЙРОННАЯ СЕТЬ

для рекомендации соотношения «матрица – наполнитель»

```
# Структура нейронной сети
best_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               1664

 dense_1 (Dense)             (None, 128)               16512

 dropout (Dropout)           (None, 128)               0

 dense_2 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 32)                2080

 dropout_2 (Dropout)         (None, 32)                0

 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 28,545
Trainable params: 28,545
Non-trainable params: 0
_________________________________________________________________
```

In [118]:
```
# Создадим функцию для генерации слоев нейронной сети
def create_NN_model(layers, activation, drop, opt):
        model= Sequential()
        for i, neurons in enumerate(layers):
            if i==0:
                model.add(Dense(neurons, input_dim=X_train_matrix.shape[1], activation = activation))  # входной слой
            else:
                model.add(Dense(neurons, activation))                                                 # добавляем полносвязный
                model.add(Dropout(drop))                                                              # исключаем переобучения
            model.add(Dense(1))                                                                       # выходной слой

            # Компиляция модели: определяем метрики и алгоритм оптимизации
            model.compile(loss = 'mse', optimizer = opt, metrics = ['mae'])

            return model
```

In [119]:
```
# Построим нейронную сети с помощью поиска по сетке с перекрестной проверкой, количество блоков равно 5 (cv = 5)
# Воспользуемся методом GridSearchCV()

reg = KerasRegressor(model = create_NN_model, layers = [128], activation = 'relu', drop = 0.1, opt = 'Adam', verbose = 2)

# Зададим параметры для модели
param_grid = {'activation': ['relu', 'softmax', 'sigmoid'],
                  'layers': [[128, 64, 16], [128, 128, 64, 32], [128, 128, 64, 16]],
                   'opt': ['Adam', 'SGD'],
                  'drop': [0.0, 0.1, 0.2],
              'batch_size': [10, 20, 40],
                  'epochs': [10, 50, 100]
             }

# Произведем поиск лучших параметров
grid = GridSearchCV(estimator = reg,
                    param_grid = param_grid,
                    cv = 5,
                    verbose = 0,
                    n_jobs = -1)

grid_result = grid.fit(X_train_matrix, np.ravel(y_train_matrix))
```

In [120]:
```
print('Лучший коэффициент R2: {:.4f} при использовании модели с параметрами {} \n'.format(grid_result.best_score_, grid_result.be
```

```
Лучший коэффициент R2: -0.0021 при использовании модели с параметрами {'activation': 'softmax', 'batch_size': 10, 'drop': 0.0,
'epochs': 10, 'layers': [128, 128, 64, 32], 'opt': 'SGD'}
```

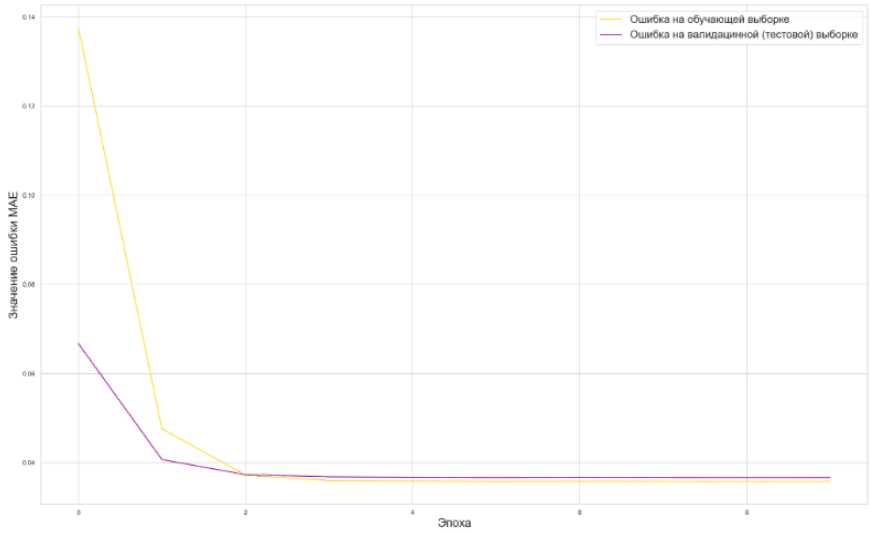
In [121]:
```
# Создадим модель с полученными значениями
best_model = Sequential()
best_model.add(Dense(128, input_dim = X_train_matrix.shape[1], activation = 'softmax'))      # входной слой
best_model.add(Dense(128, activation = 'softmax'))                                           # добавляем полносвязный слой
best_model.add(Dropout(0.0))                                                                 # исключаем переобучения
best_model.add(Dense(64, activation = 'softmax'))                                            # добавляем полносвязный слой
best_model.add(Dropout(0.0))                                                                 # исключаем переобучения
best_model.add(Dense(32, activation = 'softmax'))                                            # добавляем полносвязный слой
best_model.add(Dropout(0.0))                                                                 # исключаем переобучения
best_model.add(Dense(1))                                                                     # выходной слой

# Компиляция модели: определяем метрики и алгоритм оптимизации
best_model.compile(loss = 'mse',
                   optimizer = 'SGD',
                   metrics = ['mae'])

# Обучение модели
best_history = best_model.fit(X_train_matrix, np.ravel(y_train_matrix),
                              epochs=10,
                              batch_size=10,
                              verbose=1,
                              validation_split=0.2)
```

ОЦЕНКА МОДЕЛИ НЕЙРОННОЙ СЕТИ

График потерь на тренировочной и тестовой выборках

In [93]:
```
# Напишем функцию для построения графика потерь на тренировочной и тестовой выборок
def model_loss_plot(model_history):
    plt.figure(figsize = (25,15))
    plt.plot(model_history.history['loss'], label = 'Ошибка на обучающей выборке', color = "gold")
    plt.plot(model_history.history['val_loss'], label = 'Ошибка на валидацинной (тестовой) выборке', color = "darkmagenta")
    plt.title('График потерь модели', size=25, y=1.02)
    plt.ylabel('Значение ошибки MAE', size=20)
    plt.xlabel('Эпоха', size=20)
    plt.legend(prop={'size': 18})
    plt.show()
```

In [94]:
```
# Построим график потерь на тренировочной и тестовой выборках
model_loss_plot(best_history)
```

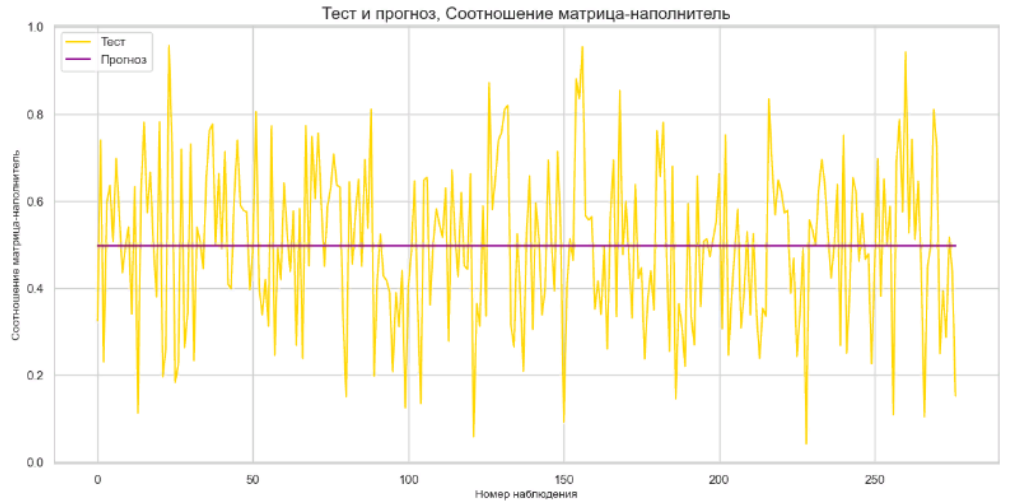


Визуализация прогнозных результатов для модели

In [95]:
```
# Создадим функцию для визуализации прогнозных результатов y_pred для модели
def Comparision_Visualization_NN(y_test, y_pred):
    plt.figure(figsize=(15,7))
    plt.title(f'Тест и прогноз, Соотношение матрица-наполнитель', size=15)
    plt.plot(np.ravel(y_test), label = 'Тест', color = "gold")
    plt.plot(y_pred, label = 'Прогноз', color = "darkmagenta")
    plt.xlabel("Номер наблюдения", size=10)
    plt.ylabel("Соотношение матрица-наполнитель", size=10)
    plt.legend(loc='best')
    plt.show()
```

In [96]:
```
Comparision_Visualization_NN(y_test_matrix, best_model.predict(X_test_matrix))
```

```
9/9 [==============================] - 0s 1ms/step
```



In [91]:
```
# Оценка получившейся модели
best_model.evaluate(X_test_matrix, np.ravel(y_test_matrix), verbose = 1)
```

```
9/9 [==============================] - 0s 4ms/step - loss: 0.0336 - mae: 0.1491
```

Out[91]:
```
[0.0336499847471714, 0.14910170435905457]
```

Вывод:

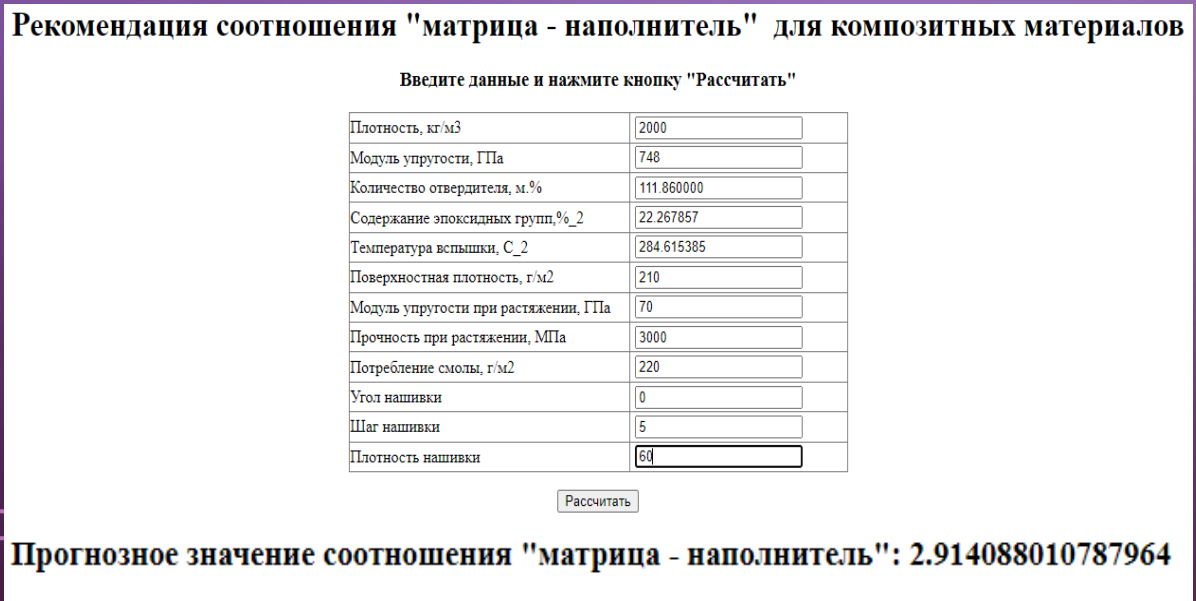
Результат прогноза нейронной сети неудовлетворительный. Значение функции потерь - среднего квадрата ошибки (MSE) - составило 0.0336, а средней абсолютной ошибки (MAE) - 0.1491.

Полученная модель нейронной сети плохо справились с поставленной задачей прогнозирования соотношения "матрица-наполнитель".

РАЗРАБОТКА ПРИЛОЖЕНИЯ

для прогнозирования соотношения «матрица – наполнитель»

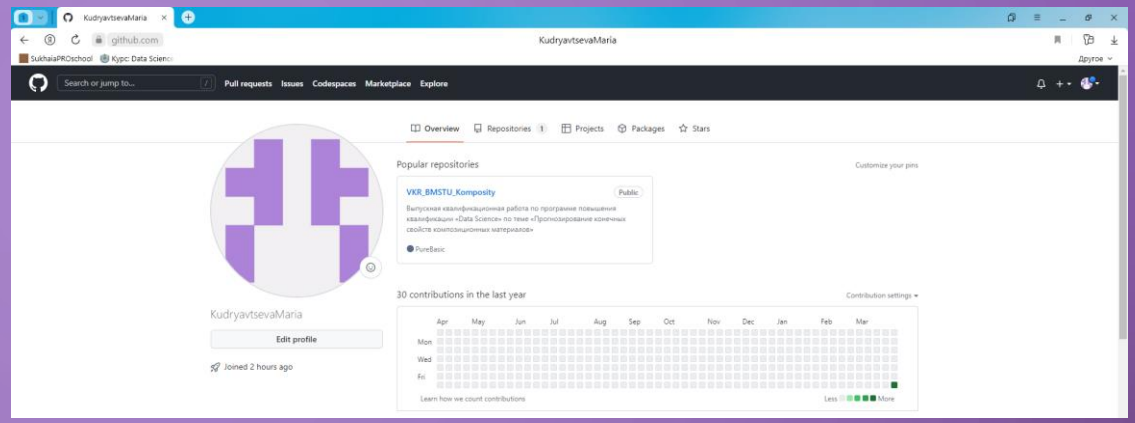
Веб-приложение в фреймворке Flask: http://127.0.0.1:5000/

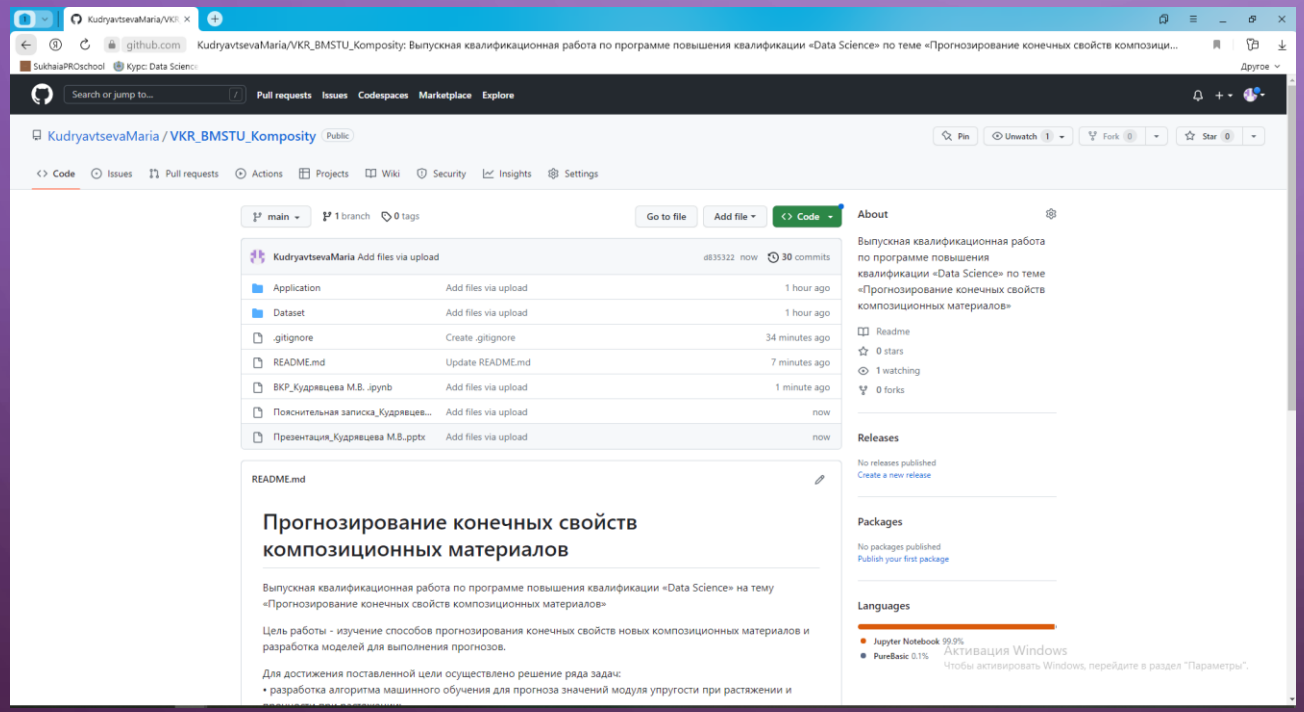


Консольное приложение

```
Приложение прогнозирует соотношение "матрица-наполнитель"
Введите "1" для прогноза, "2" для выхода
1
Введите данные для прогноза
Плотность, кг/м3: 2000
Модуль упругости, ГПа: 748
Количество отвердителя, м.%: 111.860000
Содержание эпоксидных групп,%_2: 22.267857
Температура вспышки, С_2: 284.615385
Поверхностная плотность, г/м2: 210
Модуль упругости при растяжении, ГПа: 70
Прочность при растяжении, МПа: 3000
Потребление смолы, г/м2: 220
Угол нашивки: 0
Шаг нашивки: 5
Плотность нашивки: 60
1/1 [==============================] - 0s 62ms/step
Прогнозное значение соотношения "матрица-наполнитель":
2.914088
Введите "1" для прогноза, "2" для выхода
2
```

РЕПОЗИТОРИЙ НА GITHUB





ВЫВОДЫ:

Как показал анализ исходных данных, корреляционная зависимость между характеристиками композитов крайне слабая. Этот факт непосредственно повлиял на результат работы регрессионных моделей. Все использованные модели показали низкую прогнозирующую способность.

Полученный неудовлетворительный результат может также свидетельствовать о недостатках и ошибках в наборе исходных данных, недостаточно глубокой и детальной обработке данных, неточностях в выборе алгоритмов машинного обучения и их параметров.

Для успешного решения задачи, поставленной в выпускной квалификационной работе, необходимы более глубокие знания в области материаловедения и технологии конструкционных материалов, математического анализа и статистики, а также в области решения задач машинного обучения и обработки данных. Более детальное изучение данных вопросов и консультация квалифицированных специалистов из указанных областей определенно положительно повлияют на уточнение подходов и оптимизацию алгоритмов для решения задачи прогнозирования конечных свойств композиционных материалов.

СПАСИБО ЗА ВНИМАНИЕ!