

Федеральное государственное автономное  
образовательное учреждение высшего образования  
«Пермский государственный национальный исследовательский университет»  
(ПГНИУ)  
Региональный институт непрерывного образования (РИНО ПГНИУ)  
Цифровая кафедра

Выпускная аттестационная (квалификационная) работа  
по курсу профессиональной переподготовки «Прикладной анализ данных»

**Анализ факторов, влияющих на категорию полученного осадка на  
химическом заводе**

Разработчики проекта:  
Овчинников Трофим Михайлович,  
Мазязин Максим Леонидович

Пермь, 2024

## Оглавление

ПАСПОРТ ПРОЕКТА	3
Исходные данные	4
Реализация проекта	5
Этап 1. Подготовка данных к анализу	5
Этап 2. Предварительный анализ данных	6
Этап 3. Моделирование	11
Заключение	18
Список использованных источников и литературы	19
Приложения	21

## **ПАСПОРТ ПРОЕКТА**

### **Название проекта:**

Анализ факторов, влияющих на категорию полученного осадка на химическом заводе.

### **Сведения об авторах:**

Овчинников Трофим Михайлович, Мазязин Максим Леонидович

### **Цель:**

построить классификатор, который предсказывает категорию полученного осадка (изомер №1, изомер №2, брак) по имеющимся факторным переменным.

### **Задачи:**

1. Выполнить подготовку данных к анализу.
2. Выполнить предварительный анализ данных.
3. Выполнить моделирование.
4. Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

### **Краткое описание проекта:**

Требуется проанализировать данные о химических соединениях, выявить зависимости между факторами, построить модель для прогнозирования категории полученного осадка и дать прогноз. Дать интерпретацию полученным результатам. Сделать выводы.

### **Конкретные ожидаемые результаты:**

Построить классификатор по имеющимся факторным переменным.

## Исходные данные

В предоставленном файле **train.csv** описаны параметры уже проведенных реакций и полученных результатов.

Список колонок анализируемого набора данных:

- **product** — результат реакции (*целевая переменная*):  
0 – смесь веществ (брак),  
1 – изомер продукта №1,  
2 – изомер продукта №2.
- **var\_1** — вес реагента №1 в граммах.
- **var\_2** — форма реагента №1:  
0 – порошок,  
1 – крупные гранулы.
- **var\_3** — вес реагента №2 в граммах.
- **var\_4** — форма реагента №2:  
0 – порошок,  
1 – крупные гранулы.
- **var\_5** — отклонение от температуры, указанной в инструкции (87,5°C).
- **var\_6** — материал реактора:  
0 – стекло,  
1 – металл.
- **var\_7** — отклонение от pH, указанного в инструкции (5,8).
- **var\_8** — использование механического перемешивания:  
0 – не использовалось,  
1 – использовалось.
- **var\_9** — вес катализатора реакции в граммах.
- **var\_10** — очередность загрузки реагентов в раствор:  
0 – первым загружен реагент №1,  
1 – первым загружен реагент №2.
- **var\_11** — время реакции в минутах.
- **var\_12** — вес полученного осадка в граммах.

В файле **test.csv** представлена тестовая выборка (параметры реакции без результата). В качестве решения необходимо представить прогноз категории осадка по каждой строчке файла **test.csv**.

Файл **target\_test\_example.csv** – шаблон файл с прогнозом. Именно в таком формате необходимо представить прогноз для автоматизированной проверки точности прогнозирования. Точность прогноза будет оцениваться по метрике «макро-точности»: **средняя точность по каждому классу**

## Реализация проекта

### Этап 1. Подготовка данных к анализу

Загрузим данные в датафрейм и подключим необходимые библиотеки:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
```

Загрузим данные и убедимся, что все количественные столбцы имеют числовой тип:

```
df_train = pd.read_csv('.\\train.csv', sep=';')
df_train.dtypes
```

```
product      int64
var_1        float64
var_2        int64
var_3        float64
var_4        int64
var_5        float64
var_6        int64
var_7        float64
var_8        int64
var_9        float64
var_10       int64
var_11       float64
var_12       float64
dtype: object
```

Рисунок 1. Типы данных колонок

Рассмотрим пропуски.

```
df_train.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5400 entries, 0 to 5399
Data columns (total 13 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   product     5400 non-null   int64
1   var_1       5400 non-null   float64
2   var_2       5400 non-null   int64
3   var_3       5400 non-null   float64
4   var_4       5400 non-null   int64
5   var_5       5400 non-null   float64
6   var_6       5400 non-null   int64
7   var_7       5400 non-null   float64
8   var_8       5400 non-null   int64
9   var_9       5400 non-null   float64
10  var_10      5400 non-null   int64
11  var_11      5400 non-null   float64
12  var_12      5400 non-null   float64
dtypes: float64(7), int64(6)
memory usage: 548.6 KB
```

Рисунок 2. Типы данных колонок

Итоговый датафрейм:

df\_train

	product	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12
0	0	6353.887751	0	3675.878619	0	0.301215	1	0.203161	0	0.849829	0	34.256347	1045.876061
1	0	6165.571090	0	3186.979995	1	0.510307	0	0.135070	1	0.762936	0	42.090521	968.181717
2	0	5941.886918	1	2887.991939	0	0.570525	0	0.123050	1	0.770605	0	41.458076	982.316872
3	1	5820.913609	0	3623.374054	0	0.814172	0	0.111818	0	0.976490	0	39.682286	995.101315
4	2	6022.965184	0	2850.268819	0	0.359652	0	0.231531	0	0.745337	0	37.927108	930.222551
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5395	1	7404.111197	0	3605.908115	0	0.101962	0	0.329923	1	0.714294	0	39.406563	884.975500
5396	1	5717.312110	1	3377.796681	1	0.345731	0	0.220103	0	1.012404	0	36.404537	1001.751603
5397	0	6770.863216	0	3370.264101	0	0.270617	0	0.169337	1	0.842193	0	37.314561	995.757665
5398	1	6573.093388	0	3754.622698	0	0.428129	1	0.375134	1	0.679759	0	40.673347	978.156195
5399	1	6940.983472	0	3556.359780	1	0.416299	0	0.277344	1	0.699331	1	43.866514	1126.248866

5400 rows × 13 columns

Рисунок 3. Датафрейм

## Этап 2. Предварительный анализ данных

Вычислим описательные статистики по колонкам (среднее, моду, медиану, стандартное отклонение, квантили).

*Среднее арифметическое равно сумме значений всех вариантов выборки, деленной на объем выборки:*

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n x_i.$$

Здесь  $n$  – объем выборки, а  $x_i$  – варианты выборки.

**Модой** называется значение признака, встречающееся в выборке наиболее часто. Условимся использовать для обозначения моды символы  $Mo$ .

**Медианой** является значение признака, находящееся в середине ранжированного ряда. Медиана находится по формуле

$$M_e = \begin{cases} X_{(\frac{n+1}{2})}, & \text{если } n - \text{нечетное,} \\ \frac{X_{(\frac{n}{2})} + X_{(\frac{n}{2}+1)}}{2}, & \text{если } n - \text{четное.} \end{cases}$$

**Выборочная дисперсия** находится по формуле  $S_X^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n}$ .

**Стандартным отклонением** называется положительный квадратный корень из дисперсии:

$$S = \sqrt{S^2}.$$

Оно показывает, как расположена основная часть вариант относительно среднего арифметического.

```
df_train.describe()
```

	product	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11
count	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000	5400.000000
mean	0.633704	6354.442053	0.279815	3420.197776	0.107037	0.237132	0.330741	0.224727	0.755926	0.778146	0.134259	39.030036
std	0.692813	506.846604	0.448950	272.714213	0.309189	0.306929	0.470523	0.086390	0.429577	0.088349	0.340962	3.129742
min	0.000000	4353.366793	0.000000	2391.003018	0.000000	-0.974838	0.000000	-0.079677	0.000000	0.415416	0.000000	25.338598
25%	0.000000	6011.806650	0.000000	3237.276961	0.000000	0.030203	0.000000	0.166490	1.000000	0.720105	0.000000	36.896027
50%	1.000000	6353.315711	0.000000	3419.555208	0.000000	0.239587	0.000000	0.224592	1.000000	0.778770	0.000000	39.024176
75%	1.000000	6703.425477	1.000000	3601.737154	0.000000	0.434655	1.000000	0.281945	1.000000	0.837523	0.000000	41.164110
max	2.000000	8169.159370	1.000000	4397.818747	1.000000	1.281188	1.000000	0.518510	1.000000	1.058548	1.000000	50.610822

Рисунок 4. Описательные статистики по колонкам

Проверим данные на наличие выбросов, для этого можно использовать диаграмму «ящик с усами» (boxplot). Если выбросов мало, то следует их сгладить.

График *«ящик с усами»*, или *«ящичковая диаграмма»*, или *диаграмма размаха* – график, используемый в описательной статистике и компактно изображающий одномерное распределение вероятностей. Такой вид диаграммы в удобной форме показывает медиану, нижний и верхний квартили, минимальное и максимальное значения выборки и выбросы. Данные, выходящие за границы усов (выбросы), отображаются на графике в виде точек, маленьких кружков или звёздочек. Иногда на графике отмечают среднее арифметическое и его доверительный интервал («зарубка» на ящике).

Построим отдельные диаграммы для каждой колонки:

```
fig, ax = plt.subplots(4, 3, figsize=(40, 30))

i = 0
for axi in ax:
    for axj in axi:
        axj.boxplot(df_train[df_train.columns[i]])
        axj.set_title(df_train.columns[i])
        i += 1
```

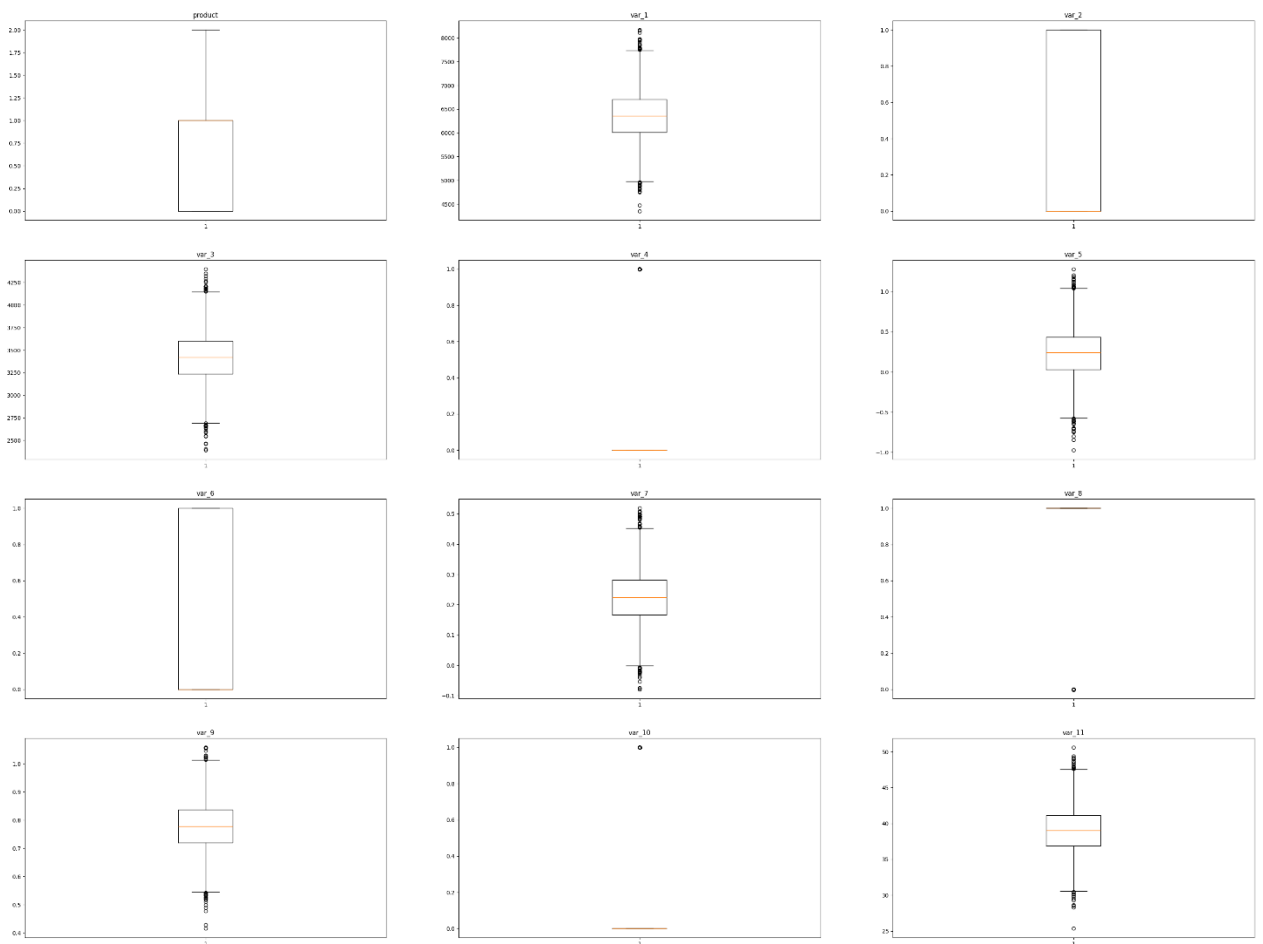


Рисунок 5. Диаграммы boxplot всех числовых колонок

Выбросов много, и они не сконцентрированы в узком диапазоне значений, а рассеяны по широкому диапазону, тогда можно ничего с ними не делать.

```
plt.rcParams["figure.figsize"] = 15, 8
i = 1
for col in df.columns[:-1]:
    plt.subplot(2, 4, i)
    plt.boxplot(df[col])
    plt.title(col)
    i += 1
plt.tight_layout();
```

Проверим данные на нормальность распределения двумя способами:

1. Построим гистограмму и сделать предположение о том, являются ли данные нормально распределенными.
2. Выполним статистический тест на нормальность и убедимся, что выдвинутое ранее предположение о нормальности верно или ошибочно.



**Гистограмма**, представляющая собой совокупность примыкающих друг к другу прямоугольников, основание каждого из которых равно ширине интервала группировки, а площадь – частоты этого интервала.

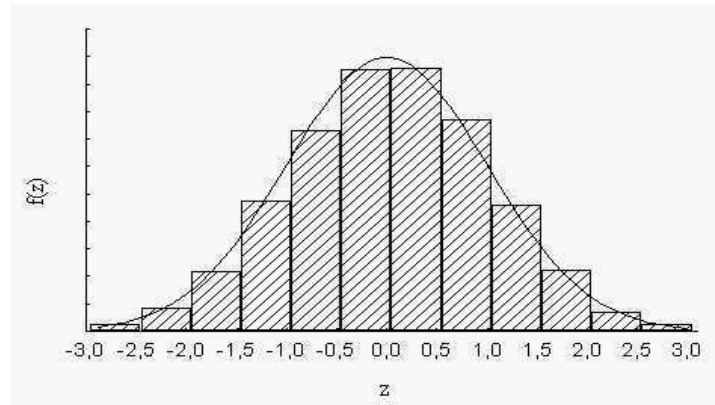


Рисунок 6. Гистограмма

Сначала построим гистограммы для всех числовых колонок:

```
fig, ax = plt.subplots(4, 3, figsize=(40, 30))
i = 0
for axi in ax:
    for axj in axi:
        axj.hist(df_train[df_train.columns[i]],
        edgecolor='k')
        axj.set_title(df_train.columns[i])
        i += 1
```

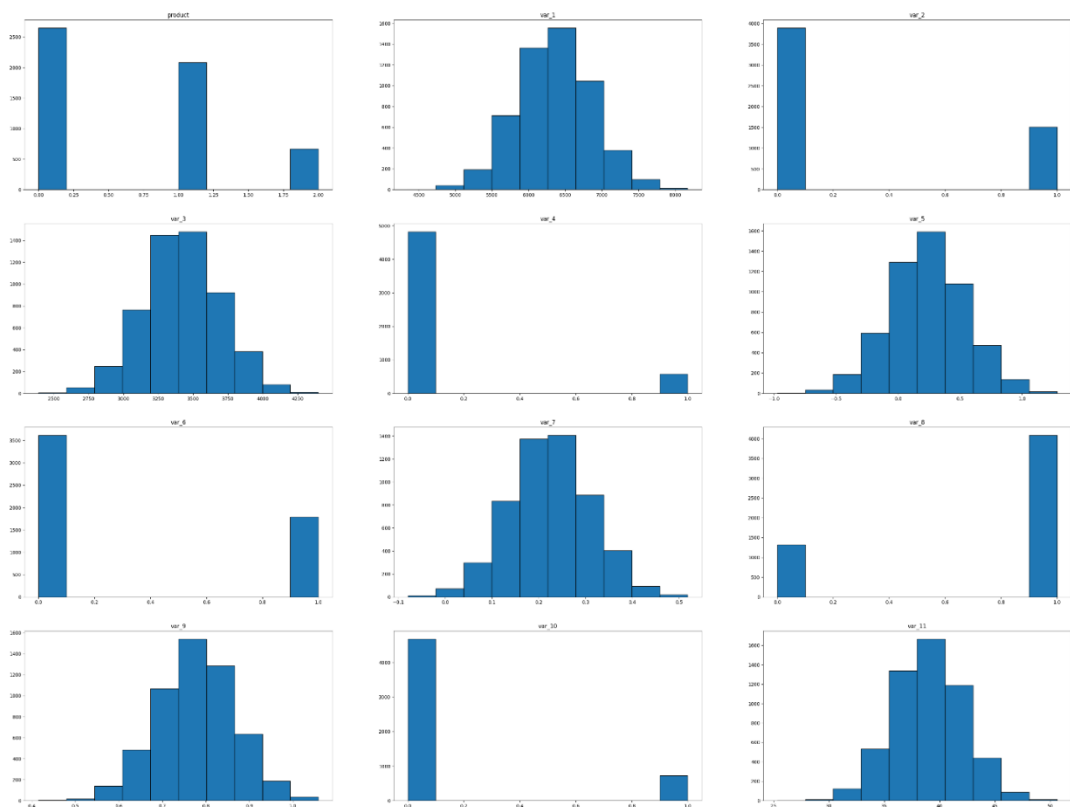


Рисунок 7. Гистограммы для всех числовых колонок

Визуально видно, что тут есть нормально распределенные колонки, но проверим через критерии согласия.

**Критерии согласия** заключаются в проверке предположения о том, что результаты наблюдений могут быть описаны с помощью определенного закона распределения (в нашем случае нормального распределения). На формальном языке проверяется гипотеза:  $H_0$  - наши данные согласуются с нормальным распределением. Если p-value меньше заданного уровня значимости (обычно 0,05 или 0,01), то основная гипотезу отвергается.

```
from scipy.stats import normaltest
print('p-values:')
for i, item in enumerate(df_train.columns):
    print(f'{item}:
          {np.round(normaltest(df_train).pvalue[i], 3)}')
```

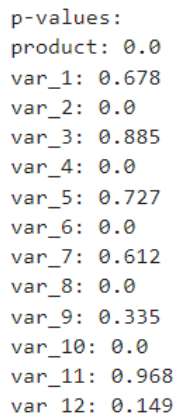


Рисунок 8. p-value для колонок

Как видим, p-value для колонок с гистограммами имеющие нормальное распределения, сделанным после визуального анализа – **подтвердилось**.

**Метод ранговой корреляции Спирмена** позволяет определить тесноту (силу) и направление корреляционной связи между двумя признаками (как количественными, так и качественными). Коэффициент ранговой корреляции имеет границы изменения от  $-1$  до  $+1$ . Полное совпадение рангов означает максимально тесную прямую связь, полная противоположность рангов – максимально тесную обратную связь.

Матрицу корреляции отобразим с помощью диаграммы «тепловая карта» (heatmap):

```
import seaborn as sns
sns.heatmap(df_train.corr())
```

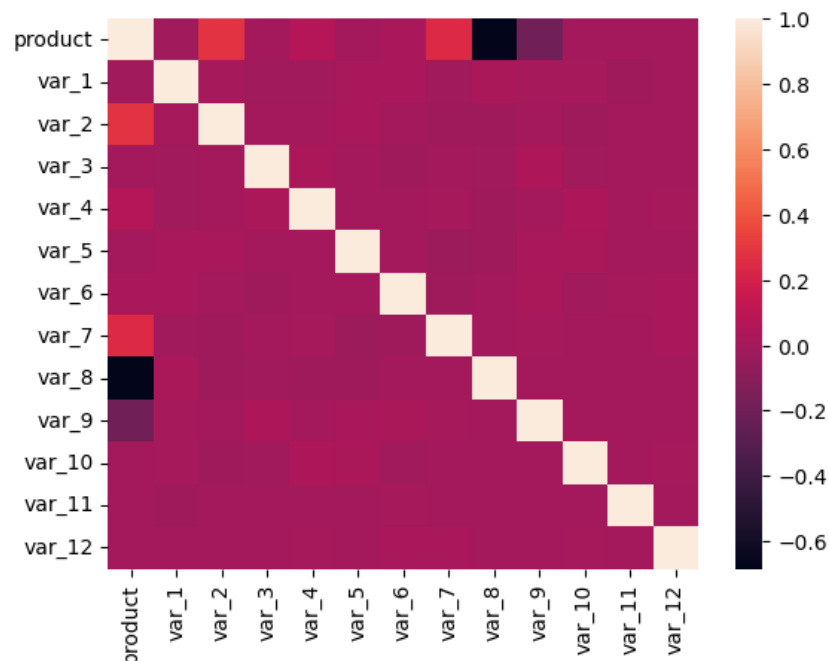


Рисунок 9. Тепловая карта матрицы корреляции

Выполнить нормализацию числовых столбцов – факторных признаков путем приведения их значений к диапазону от 0 до 1.

```
df_scl = df_train.copy()
df_scl[df_scl.columns[1:]]=(df_scl[df_scl.columns[1:]] -
df_scl[df_scl.columns[1:]].min())/(df_scl[df_scl.columns[
1:]].max() - df_scl[df_scl.columns[1:]].min())
df_scl
```

	product	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12
0	0	0.524274	0.0	0.640256	0.0	0.565620	1.0	0.472826	0.0	0.675465	0.0	0.352868	0.582686
1	0	0.474922	0.0	0.396637	1.0	0.658301	0.0	0.358997	1.0	0.540356	0.0	0.662859	0.452190
2	0	0.416301	1.0	0.247651	0.0	0.684994	0.0	0.338902	1.0	0.552279	0.0	0.637834	0.475932
3	1	0.384598	0.0	0.614093	0.0	0.792992	0.0	0.320125	0.0	0.872408	0.0	0.567567	0.497405
4	2	0.437550	0.0	0.228853	0.0	0.591523	0.0	0.520253	0.0	0.512991	0.0	0.498116	0.388434
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5395	1	0.799505	0.0	0.605389	0.0	0.477300	0.0	0.684735	1.0	0.464723	0.0	0.556657	0.312437
5396	1	0.357447	1.0	0.491721	1.0	0.585352	0.0	0.501148	0.0	0.928252	0.0	0.437870	0.508574
5397	0	0.633550	0.0	0.487968	0.0	0.552057	0.0	0.416282	1.0	0.663591	0.0	0.473878	0.498507
5398	1	0.581721	0.0	0.679494	0.0	0.621876	1.0	0.760316	1.0	0.411024	0.0	0.606783	0.468943
5399	1	0.678133	0.0	0.580699	1.0	0.616632	0.0	0.596839	1.0	0.441457	1.0	0.733134	0.717681

Рисунок 10. Нормализованный датафрейм

### Этап 3. Моделирование

Разбиваем данные на обучающую и тестовую выборки (соотношение выбрать самостоятельно, обучающая выборка должна быть больше), предварительно перемешав:

```
df_test = pd.read_csv('..\test.csv', sep=';')
df_test = (df_test - df_test.min()) / (df_test.max() -
df_test.min())
df_test
```

	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12
0	0.308016	0.0	0.626910	0.0	0.791844	0.0	0.590004	1.0	0.595869	0.0	0.359665	0.577201
1	0.356556	0.0	0.496608	0.0	0.422435	0.0	0.601545	1.0	0.409507	0.0	0.417239	0.712693
2	0.718715	0.0	0.510183	0.0	0.187158	1.0	0.545514	0.0	0.262117	0.0	0.462337	0.362056
3	0.791983	0.0	0.570453	0.0	0.722101	1.0	0.723881	1.0	0.585307	0.0	0.484365	0.389547
4	0.610155	0.0	0.076976	1.0	0.000000	0.0	0.362181	0.0	0.553509	1.0	0.428099	0.522723
...	...	...	...	...	...	...	...	...	...	...	...	...
595	0.585898	1.0	0.667731	0.0	0.417486	0.0	0.635316	1.0	0.428451	0.0	0.555615	0.397090
596	0.716314	1.0	0.298439	0.0	0.873416	1.0	0.453192	1.0	0.583785	0.0	0.009939	0.450338
597	0.652965	0.0	0.636648	0.0	0.267032	0.0	0.534777	1.0	0.416602	1.0	0.386781	0.628088
598	0.618075	0.0	0.455552	0.0	0.374092	1.0	0.521543	0.0	0.506461	0.0	0.437634	0.388867
599	0.772312	0.0	0.623347	1.0	0.372659	1.0	0.731089	1.0	0.305574	0.0	0.263165	0.753837

600 rows × 12 columns

#### Рисунок 11. Тестовый датафрейм

На обучающей выборке построим несколько моделей, применив различные алгоритмы классификации:

```
from sklearn.model_selection import train_test_split
df1, df2 = train_test_split(df_scl)

X = df1[df1.columns[1:]]
y = df1[df1.columns[0]]

X_test = df2[df2.columns[1:]]
y_test = df2[df2.columns[0]]

f1_scores = []
```

#### 1. SVM:

```
from sklearn import svm
svm_clf = svm.SVC()
svm_clf.fit(X, y)
```

```

from sklearn.metrics import f1_score
f1_scores.append(['SVM', f1_score(y_test,
svm_clf.predict(X_test), average='micro')])

```

## 2. Stochastic Gradient Descent:

```

from sklearn.linear_model import SGDCClassifier as SGDC
sgdc_clf = SGDC(loss="hinge", penalty="l2", max_iter=100)
sgdc_clf.fit(X, y)
f1_scores.append(['Stochastic Gradient Descent',
f1_score(y_test, sgdc_clf.predict(X_test),
average='micro')])

```

## 3. KNeighborsClassifier:

```

from sklearn.neighbors import KNeighborsClassifier as KNC
knc_clf = KNC(n_neighbors=5)
knc_clf.fit(X, y)
f1_scores.append(['KNeighborsClassifier', f1_score(y_test,
knc_clf.predict(X_test), average='micro')])

```

## 4. RadiusNeighborsClassifier:

```

from sklearn.neighbors import RadiusNeighborsClassifier as RNC
rnc_clf = RNC()
rnc_clf.fit(X, y)
f1_scores.append(['RadiusNeighborsClassifier',
f1_score(y_test, rnc_clf.predict(X_test), average='micro')])

```

## 5. GaussianProcessClassifier:

```

from sklearn.gaussian_process import
GaussianProcessClassifier as GPC
gpc_clf = GPC()
gpc_clf.fit(X, y)
f1_scores.append(['GaussianProcessClassifier',
f1_score(y_test, gpc_clf.predict(X_test), average='micro')])

```

## 6. DecisionTreeClassifier:

```

from sklearn.tree import DecisionTreeClassifier as DTC
dtc_clf = DTC()
dtc_clf.fit(X, y)
f1_scores.append(['DecisionTreeClassifier', f1_score(y_test,
dtc_clf.predict(X_test), average='micro')])

```

## 7. AdaBoostClassifier:

```

from sklearn.ensemble import AdaBoostClassifier as ABC
abc_clf = ABC()
abc_clf.fit(X, y)
f1_scores.append(['AdaBoostClassifier', f1_score(y_test,
abc_clf.predict(X_test), average='micro')])

```

#### 8. BaggingClassifier:

```
from sklearn.ensemble import BaggingClassifier as bag
bag_clf = bag()
bag_clf.fit(X, y)
f1_scores.append(['BaggingClassifier', f1_score(y_test,
bag_clf.predict(X_test), average='micro')])
```

#### 9. ExtraTreesClassifier:

```
from sklearn.ensemble import ExtraTreesClassifier as ETC
etc_clf = ETC()
etc_clf.fit(X, y)
f1_scores.append(['ExtraTreesClassifier', f1_score(y_test,
etc_clf.predict(X_test), average='micro')])
```

#### 10. GradientBoostingClassifier:

```
from sklearn.ensemble import GradientBoostingClassifier as GBC
gbc_clf = GBC()
gbc_clf.fit(X, y)
f1_scores.append(['GradientBoostingClassifier',
f1_score(y_test, gbc_clf.predict(X_test), average='micro')])
```

#### 11. RandomForestClassifier:

```
from sklearn.ensemble import RandomForestClassifier as RFC
rfc_clf = RFC()
rfc_clf.fit(X, y)
f1_scores.append(['RandomForestClassifier', f1_score(y_test,
rfc_clf.predict(X_test), average='micro')])
```

Статистика критерия (F1-мера) вычисляется по формуле

$$F1 = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Максимально возможное значение F-показателя равно 1,0, что указывает на идеальную точность и полноту выборки, а минимально возможное значение равно 0, если точность или полнота выборки равны нулю.

Вычислим показатель качества классификации F1-мера на тестовой выборке:

```
f1_scores
```

```
[['SVM', np.float64(0.8644444444444445)],
 ['Stochastic Gradient Descent', np.float64(0.825925925925926)],
 ['KNeighborsClassifier', np.float64(0.8162962962962963)],
 ['RadiusNeighborsClassifier', np.float64(0.6874074074074074)],
 ['GaussianProcessClassifier', np.float64(0.8555555555555555)],
 ['DecisionTreeClassifier', np.float64(0.7992592592592592)],
 ['AdaBoostClassifier', np.float64(0.6555555555555556)],
 ['BaggingClassifier', np.float64(0.8725925925925926)],
 ['ExtraTreesClassifier', np.float64(0.8837037037037037)],
 ['GradientBoostingClassifier', np.float64(0.8888888888888888)],
 ['RandomForestClassifier', np.float64(0.8896296296296297)]]
```

Рисунок 12. Результаты f1\_scores

Результат: F1-мера у всех методов стремиться к 1, ближе всего к единице **RandomForestClassifier**.

Используя 2 числовых фактора в качестве координат, строим диаграмму рассеяния на тестовой выборке, при этом, верно, классифицированные един. объекты обозначить закрашенным кружком, верно классифицированные нул. объекты – незакрашенным кружком, неверно классифицированные объекты – крестиком:

```
X0True = X_test[(y_test == rfc_clf.predict(X_test)) *
(y_test == 0)]
X1True = X_test[(y_test == rfc_clf.predict(X_test)) *
(y_test == 1)]
X2True = X_test[(y_test == rfc_clf.predict(X_test)) *
(y_test == 2)]
XFalse = X_test[(y_test != rfc_clf.predict(X_test))]

plt.scatter(X0True.var_9, X0True.var_11, marker='o',
label='Истинные 0')
plt.scatter(X1True.var_9, X1True.var_11, marker='>',
label='Истинные 1')
plt.scatter(X2True.var_9, X2True.var_11, marker='<',
label='Истинные 2')
plt.scatter(XFalse.var_9, XFalse.var_11, marker='x',
label='Ложные')

plt.legend()
plt.xlabel('Вес катализатора реакции')
```

```
plt.ylabel('Время реакции')
```

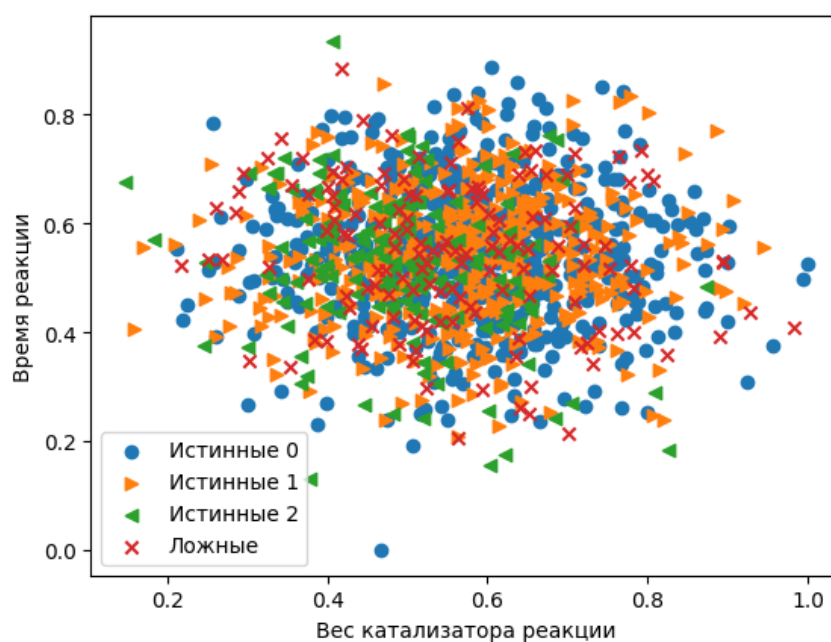


Рисунок 13. Диаграмма рассеяния

Вычислить другие показатели качества классификации: ассигасу, precision, recall, построить матрицу ошибок:

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test,
rfc_clf.predict(X_test)))
```

	precision	recall	f1-score	support
0	0.91	0.91	0.91	674
1	0.87	0.83	0.85	506
2	0.88	0.99	0.93	170
accuracy			0.89	1350
macro avg	0.88	0.91	0.90	1350
weighted avg	0.89	0.89	0.89	1350

Рисунок 10. Показатели качества классификации

```
sns.heatmap(confusion_matrix(y_test, rfc_clf.predict
(X_test)))
```



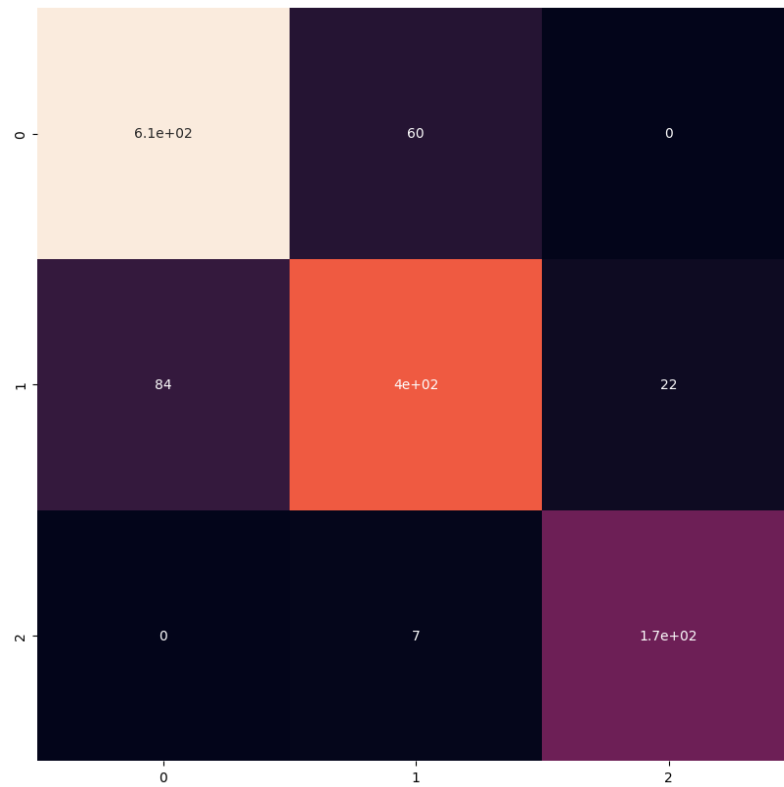


Рисунок 14. Матрица ошибок

`rfc_clf.predict(df_test)`

```
array([0, 1, 2, 1, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 1, 1, 0, 0, 1, 2, 1, 0,
       1, 1, 0, 0, 0, 1, 2, 0, 2, 0, 1, 1, 0, 1, 1, 1, 0, 2, 1, 0, 0, 0,
       1, 2, 1, 1, 1, 1, 2, 0, 1, 2, 1, 2, 0, 2, 1, 0, 0, 0, 1, 0, 1, 0,
       2, 0, 2, 1, 0, 2, 0, 2, 1, 1, 0, 1, 2, 2, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 2, 0, 1, 0, 2, 0, 1, 2, 1, 1, 0, 0, 1, 0, 0, 0, 2, 2,
       0, 0, 2, 1, 1, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 2, 1, 1,
       0, 0, 2, 1, 1, 2, 1, 0, 1, 0, 0, 2, 2, 0, 2, 0, 0, 2, 2, 1, 0, 1,
       0, 2, 0, 0, 1, 0, 2, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 2, 1, 1,
       0, 1, 0, 2, 1, 1, 2, 0, 2, 0, 0, 1, 0, 0, 2, 1, 0, 2, 0, 1, 1, 1,
       1, 1, 0, 2, 1, 1, 1, 0, 2, 0, 1, 1, 0, 1, 2, 1, 0, 0, 1, 1, 1,
       0, 2, 0, 2, 1, 2, 0, 0, 2, 1, 1, 2, 1, 1, 0, 0, 0, 1, 2, 1, 0, 0,
       2, 0, 2, 0, 0, 2, 0, 1, 0, 1, 1, 0, 0, 0, 2, 0, 1, 0, 0, 2, 1, 1,
       0, 0, 1, 2, 1, 1, 1, 1, 1, 1, 1, 0, 0, 2, 2, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 2, 1, 0, 0, 0, 2, 0, 0, 2, 2, 1, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 0, 2, 1, 0, 1, 1, 2, 1, 1, 0, 1, 1, 1, 1, 0, 2, 1, 2, 0, 1,
       0, 1, 0, 1, 0, 0, 2, 2, 1, 0, 2, 0, 0, 0, 0, 0, 2, 0, 1, 1, 2, 1,
       0, 0, 2, 2, 0, 1, 2, 1, 0, 0, 1, 0, 0, 2, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 2, 2, 2, 1, 1, 1, 1, 2, 1, 0, 0, 2, 1, 0, 2, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 2, 0, 2, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 2, 2,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 1, 2, 0, 1, 1, 2, 1,
       0, 2, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 1, 2, 0, 0, 1, 2, 0, 1, 0, 0, 1,
       2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 2, 0, 2, 2, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 2, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 2, 0, 2, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 2, 1, 1, 2, 0, 0, 1, 2, 0, 1,
       2, 1, 0, 0, 2, 0, 2, 1, 1, 0, 0, 1, 2, 0, 1, 0, 0, 0, 2, 0, 0,
       1, 1, 0, 0, 2, 1])
```

Рисунок 15. Прогнозируемый результат для файла ‘test.csv’

## Заключение

Задачи работы были выполнены в то числе:

1. Выполнена подготовка данных к анализу (рис.3).
2. Выполнен предварительный анализ данных (рис. 5, 7, 9, 10).
3. Выполнено моделирование (SVM, Stochastic Gradient Descent, KNeighborsClassifier, RadiusNeighborsClassifier, GaussianProcessClassifier, DecisionTreeClassifier, AdaBoostClassifier, BaggingClassifier, ExtraTreesClassifier, GradientBoostingClassifier, RandomForestClassifier) и найдена f1-мера для них (рис. 12).

Цель работы была выполнена построить классификатор, который предсказывает категорию полученного осадка (изомер №1, изомер №2, брак) по имеющимся факторным переменным - построена диаграмма рассеяния (рис. 13) и прогнозируемый результат (рис. 14)

## Список использованных источников и литературы

1. Федин, Ф. О. Анализ данных. Часть 1. Подготовка данных к анализу : учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва : Московский городской педагогический университет, 2012. — 204 с. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
2. Амоа К.А. Разработка программных пакетов на языке Python: учебное пособие / К.А. Амоа, Н.А. Рындин, Ю.С. Скворцов. – Воронеж: Воронежский государственный технический университет, ЭБС АСВ, 2020. – 61 с. // Электронно-библиотечная система IPR BOOKS: [сайт].
3. Гуриков С. Р. Основы алгоритмизации и программирования на Python : учеб. пособие / С.Р. Гуриков. ? М. : ФОРУМ : ИНФРА-М, 2018. ? 343 с. ? (Высшее образование: Бакалавриат). - Режим доступа: <http://znanium.com/catalog/product/924699>
4. Вандер П. Python для сложных задач: наука о данных и машинное обучение. — СПб.: Питер, 2018. — 576 с.
5. Васильев, А.Н. Программирование на Python в примерах и задачах / А.Н. Васильев. — Москва : Эксмо, 2021. — 616 с.
6. Сузи Р.А. Язык программирования Python: учебное пособие / Р.А. Сузи. — 3-е изд. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 350 с. // Электронно-библиотечная система IPR BOOKS: [сайт].
7. Волкова В.М., Программные системы статистического анализа. Обнаружение закономерностей в данных с использованием системы R и языка Python [Электронный ресурс]: учебное пособие / Волкова В.М. - Новосибирск : Изд-во НГТУ, 2017. - 74 с. - ISBN 978-5-7782-3183-2 - Режим доступа: <http://www.studentlibrary.ru/book/ISBN9785778231832.html>
8. Криволапов С.Я. Математика на Python : учебник / С.Я. Криволапов, М.Б. Хрипунова. — Москва: КНОРУС, 2022. — 456 с.

9. Федин, Ф. О. Анализ данных. Часть 1. Подготовка данных к анализу : учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва : Московский городской педагогический университет, 2012. — 204 с. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
10. Федин, Ф. О. Анализ данных. Часть 2. Инструменты DataMining : учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва : Московский городской педагогический университет, 2012. — 308 с. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
11. Федоров Д.Ю. Программирование на языке высокого уровня Python: учебное пособие / Д.Ю. Федоров. – 2-е изд.– М.: Юрайт, 2020. – 161 с.
12. Филлипс Т. Управление на основе данных. Как интерпретировать цифры и принимать качественные решения в бизнесе. – М.:Манн, Иванов и Фербер, 2017. - 117 с.
13. Фрэнкс, Б. Революция в аналитике: как в эпоху BigData улучшить ваш бизнес с помощью операционной аналитики / Б. Фрэнкс; Пер. с англ. И. Евстигнеевой; Ред. В. Мылов. – М.: Альпина Паблишер, 2016. – 315 с.

### Программный код

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk

df_train = pd.read_csv('.\\train.csv', sep=';')
df_train.dtypes

df_train.info()

df_train

df_train.describe()

fig, ax = plt.subplots(4, 3, figsize=(40, 30))

i = 0
for axi in ax:
    for axj in axi:
        axj.boxplot(df_train[df_train.columns[i]])
        axj.set_title(df_train.columns[i])
        i += 1

plt.rcParams["figure.figsize"] = 15, 8
i = 1
for col in df.columns[:-1]:
    plt.subplot(2, 4, i)
    plt.boxplot(df[col])
    plt.title(col)
    i += 1
plt.tight_layout();

from scipy.stats import normaltest
print('p-values:')
for i, item in enumerate(df_train.columns):
    print(f'{item}:
        {np.round(normaltest(df_train).pvalue[i], 3)}')

df_test = pd.read_csv('.\\test.csv', sep=';')
df_test = (df_test - df_test.min()) / (df_test.max() -
df_test.min())
df_test
```

```

from sklearn.model_selection import train_test_split
df1, df2 = train_test_split(df_scl)

X = df1[df1.columns[1:]]
y = df1[df1.columns[0]]

X_test = df2[df2.columns[1:]]
y_test = df2[df2.columns[0]]

f1_scores = []

from sklearn import svm
svm_clf = svm.SVC()
svm_clf.fit(X, y)
from sklearn.metrics import f1_score
f1_scores.append(['SVM', f1_score(y_test,
svm_clf.predict(X_test), average='micro')])

from sklearn.linear_model import SGDClassifier as SGDC
sgdc_clf = SGDC(loss="hinge", penalty="l2", max_iter=100)
sgdc_clf.fit(X, y)
f1_scores.append(['Stochastic Gradient Descent',
f1_score(y_test, sgdc_clf.predict(X_test),
average='micro')])

from sklearn.neighbors import KNeighborsClassifier as KNC
knc_clf = KNC(n_neighbors=5)
knc_clf.fit(X, y)
f1_scores.append(['KNeighborsClassifier', f1_score(y_test,
knc_clf.predict(X_test), average='micro')])

from sklearn.neighbors import RadiusNeighborsClassifier as RNC
rnc_clf = RNC()
rnc_clf.fit(X, y)
f1_scores.append(['RadiusNeighborsClassifier',
f1_score(y_test, rnc_clf.predict(X_test), average='micro')])

from sklearn.gaussian_process import
GaussianProcessClassifier as GPC
gpc_clf = GPC()
gpc_clf.fit(X, y)
f1_scores.append(['GaussianProcessClassifier',
f1_score(y_test, gpc_clf.predict(X_test), average='micro')])

from sklearn.tree import DecisionTreeClassifier as DTC
dtc_clf = DTC()
dtc_clf.fit(X, y)
f1_scores.append(['DecisionTreeClassifier', f1_score(y_test,
dtc_clf.predict(X_test), average='micro')])

from sklearn.ensemble import AdaBoostClassifier as ABC

```

```

abc_clf = ABC()
abc_clf.fit(X,y)
f1_scores.append(['AdaBoostClassifier', f1_score(y_test,
abc_clf.predict(X_test), average='micro'))])

from sklearn.ensemble import BaggingClassifier as bag
bag_clf = bag()
bag_clf.fit(X, y)
f1_scores.append(['BaggingClassifier', f1_score(y_test,
bag_clf.predict(X_test), average='micro'))])

from sklearn.ensemble import ExtraTreesClassifier as ETC
etc_clf = ETC()
etc_clf.fit(X, y)
f1_scores.append(['ExtraTreesClassifier', f1_score(y_test,
etc_clf.predict(X_test), average='micro'))])

from sklearn.ensemble import GradientBoostingClassifier as
GBC
gbc_clf = GBC()
gbc_clf.fit(X, y)
f1_scores.append(['GradientBoostingClassifier',
f1_score(y_test, gbc_clf.predict(X_test), average='micro'))])

from sklearn.ensemble import RandomForestClassifier as RFC
rfc_clf = RFC()
rfc_clf.fit(X,y)
f1_scores.append(['RandomForestClassifier', f1_score(y_test,
rfc_clf.predict(X_test), average='micro'))])

f1_scores

X0True = X_test[(y_test == rfc_clf.predict(X_test)) *
(y_test == 0)]
X1True = X_test[(y_test == rfc_clf.predict(X_test)) *
(y_test == 1)]
X2True = X_test[(y_test == rfc_clf.predict(X_test)) *
(y_test == 2)]
XFalse = X_test[(y_test != rfc_clf.predict(X_test))]
plt.scatter(X0True.var_9, X0True.var_11, marker='o',
label='Истинные 0')
plt.scatter(X1True.var_9, X1True.var_11, marker='>',
label='Истинные 1')
plt.scatter(X2True.var_9, X2True.var_11, marker='<',
label='Истинные 2')

```

```

plt.scatter(XFalse.var_9, XFalse.var_11, marker='x',
label='Ложные')
plt.legend()
plt.xlabel('Вес катализатора реакции')
plt.ylabel('Время реакции')
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test,
rfc_clf.predict(X_test)))

sns.heatmap(confusion_matrix(y_test, rfc_clf.predict
(X_test)))

rfc_clf.predict(df_test)

```