Annot

задача:

> Химическая промышленность: Веб-приложение, которое на основе ИИ будет используется для отслеживания химического состава природных водных ресурсов. Оно будет анализировать изменения в составе воды и предоставлять рекомендации для минимизации воздействия на окружающую среду и на здоровье человека.

Ввод [1]:

```
import os
import pandas as pd
import numpy as np
import pickle

from typing import List

import seaborn as sns
from matplotlib import pylab as plt
%matplotlib inline
plt.style.use('bmh')
plt.rcParams.update({"font.size": 14})
```

Данные + EDA

Данные найдены на Kaggle (https://www.kaggle.com/datasets/adityakadiwal/water-potability (https://www.kaggle.com/datasets/adityakadiwal/water-potability))

Ввод [2]:

```
data = pd.read_csv("../data/water_potability.csv")
print(f"Датасет содержит {data.shape[0]} строк и {data.shape[1]} столбцов")
init_columns = data.columns
```

```
Датасет содержит 3276 строк и 10 столбцов
```

Ввод [3]:

```
data.sample()
```

Out[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
2259	7.855906	209.751802	19850.182892	8.038872	300.292954	368.437561	18.525207	57.238874	4.15189	0

> Определим пропущенные данные

Ввод [4]:

```
for col in data.columns:
    print(f"{col:30s} :: {str(data[col].isna().sum()):4s} {data[col].isna().sum() / data.shape[0] * 100:.0f
```

```
ph                             :: 491  15%
Hardness                       :: 0    0%
Solids                         :: 0    0%
Chloramines                    :: 0    0%
Sulfate                        :: 781  24%
Conductivity                   :: 0    0%
Organic_carbon                 :: 0    0%
Trihalomethanes                :: 162  5%
Turbidity                      :: 0    0%
Potability                     :: 0    0%
```

> Так как предполагается работа в продакшене, используем дополнительные фичи для маркировки пропусков в данных. Более того, подобный подход позволит обучить более устойчивую модель.

Ввод [5]:

```
for col in data.columns:
    if col != "Potability":
        data[f"{col}_nan"] = data[col].isna().astype(int)
        data.loc[data[col].isna(), col] = 0.0
data.sample()
```

Out[5]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability	ph_n
1020	5.21899	164.790992	27007.886906	6.592517	0.0	422.223802	14.070879	73.92032	4.383943	0	

Ввод [6]:

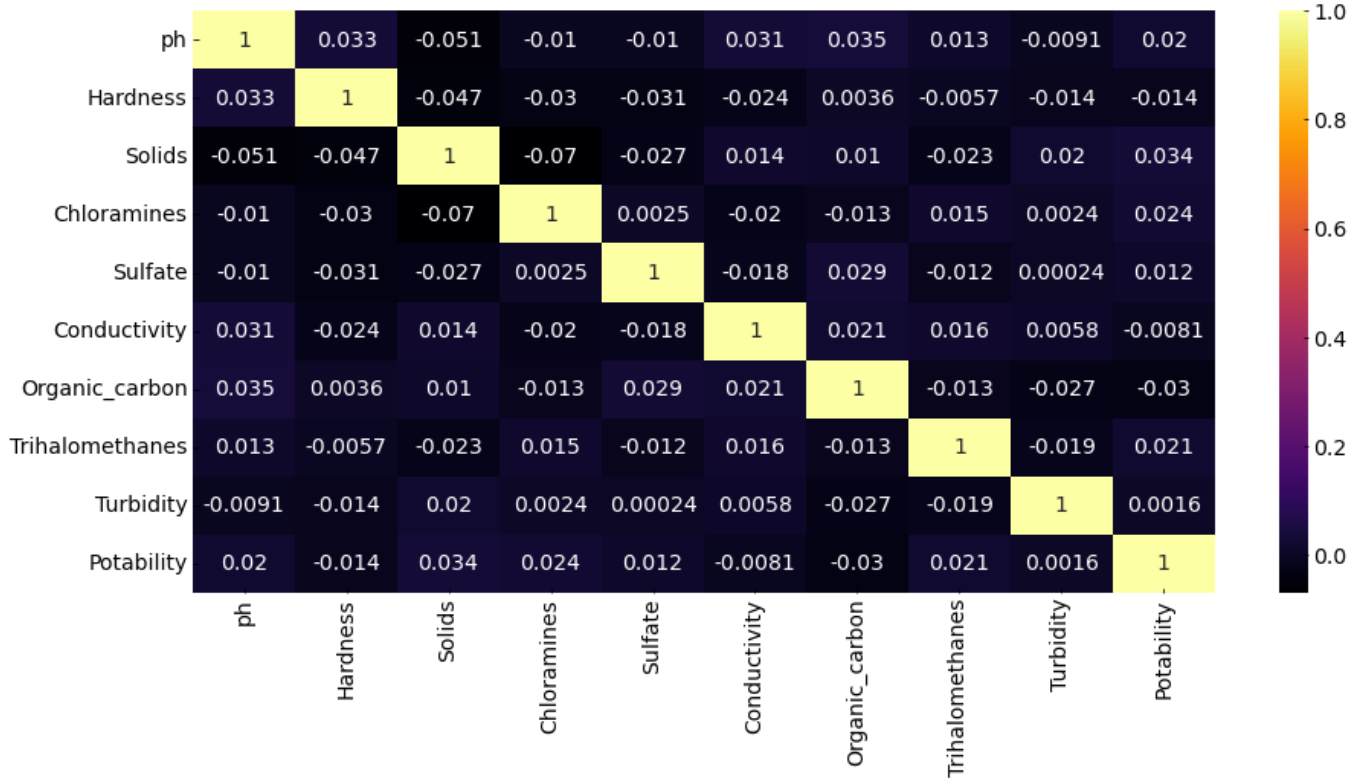
```
print(f"Пропусков в обогащенных данных: {data.isna().sum().sum()}")
```

```
Пропусков в обогащенных данных: 0
```

Посмотрим на корреляции между целевой переменной и исходными колонками

Ввод [7]:

```
corr = data[init_columns].corr()
plt.figure(figsize=(15, 7))
sns.heatmap(corr, cmap="inferno", annot=True)
plt.show()
```

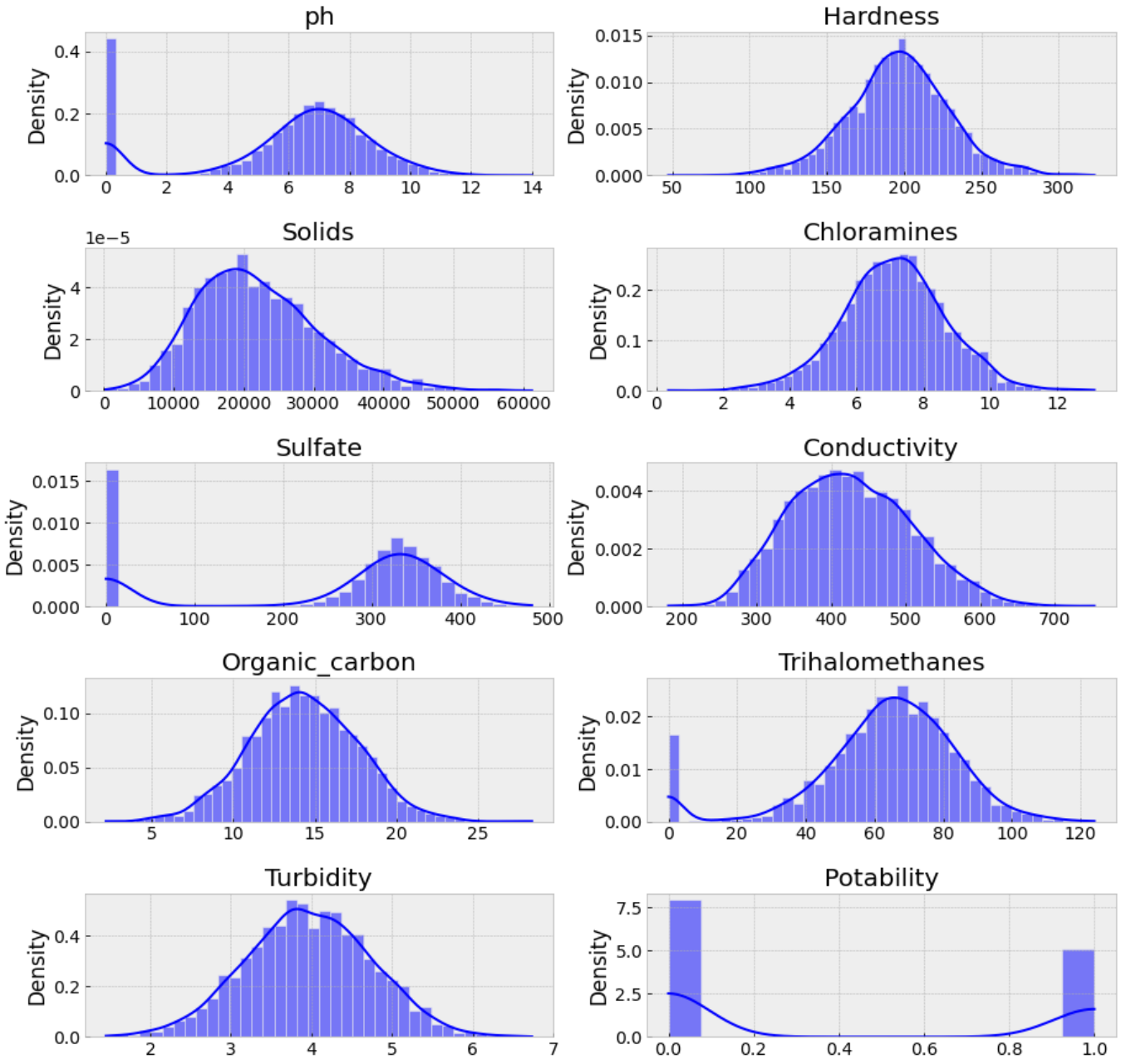


Данные достаточно чистые, в плане внутренних корреляций. Нет сильно коррелированных фичей.

Визуализируем распределения значений

Ввод [8]:

```
fig, ax = plt.subplots(5, 2, figsize=(15, 15))

for idx, col in enumerate(init_columns):
    sns.histplot(data[col], ax=ax[idx // 2, idx % 2], kde=True, stat='density')
    ax[idx // 2, idx % 2].set_title(col)
    ax[idx // 2, idx % 2].set_xlabel("")
plt.subplots_adjust(hspace=0.5)
```



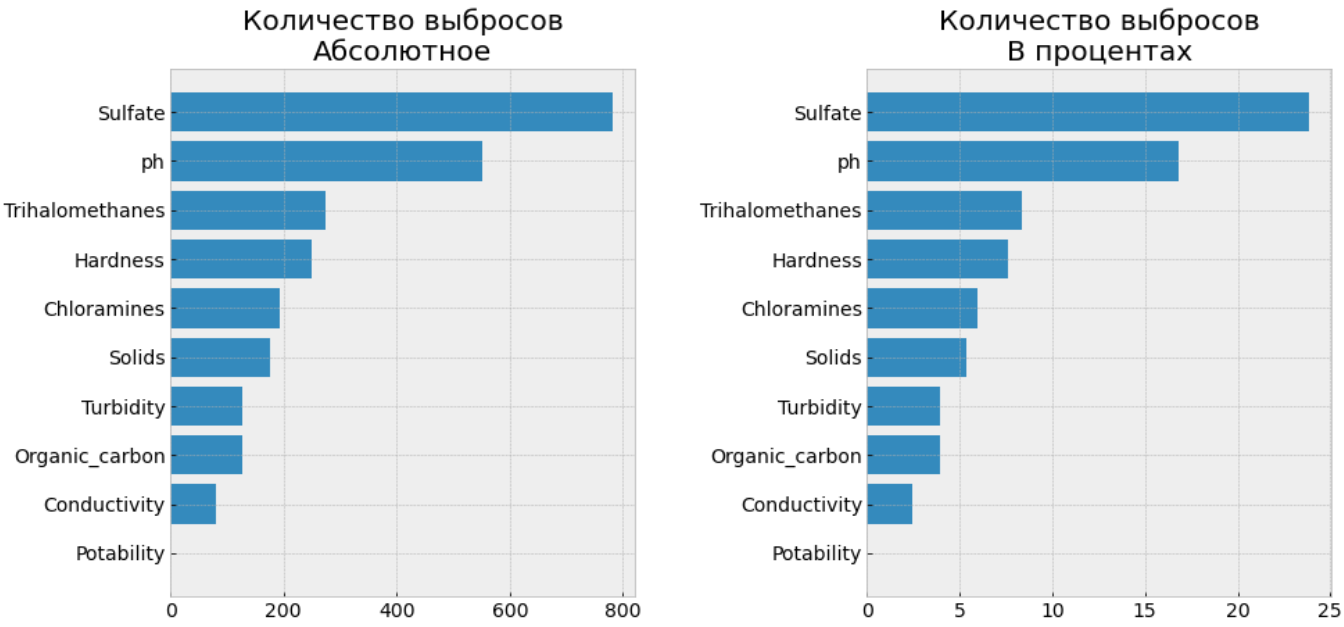
Данные распределены близко к нормальному распределению. Только `Solids` немного смещен. Проверим на выбросы. Нули - замещенные NaN

Ввод [9]:

```
def get_outliners(x: np.ndarray) -> int:
    """Реализация выявления выбросов, которые не укладываются в 1.5 * IQR

    Args:
        x (np.ndarray): входной набор данных (вектор)

    Returns:
        (int): количество выбросов
    """
    iqr = 1.5 * (np.quantile(x, 0.75) - np.quantile(x, 0.25))
    low_bond = x.median() - iqr
    upper_bound = x.median() + iqr
    return np.where(x > upper_bound)[0].size + np.where(x < low_bond)[0].size
```

Ввод [10]:

```
_temp_series = pd.Series({col: get_outliners(data[col]) for col in init_columns})
_temp_series = _temp_series.sort_values()

fig, ax = plt.subplots(1, 2, figsize=(15, 7))
ax[0].barh(_temp_series.index, _temp_series.values)
ax[0].set_title("Количество выбросов\nАбсолютное")
ax[1].barh(_temp_series.index, _temp_series.values / len(data) * 100)
ax[1].set_title("Количество выбросов\nВ процентах")
plt.subplots_adjust(wspace=0.5)
plt.show()
```



> Так как выбросов либо нет, либо они не единичные (от 2.5 % значений фичи) можно заключить, что это не выбросы, а **особенности данных**

Train

Будем учить XGBoost [на GPU, чтоб побыстрее, в инференсе будет CPU]. Параметры найдем перебором.

План обучения:

- baseline на чистом датасете
- повышение устойчивости путем генерации пропусков в данных

Ввод [11]:

```
import time
import pickle

from xgboost import XGBClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, recall_score, roc_auc_score, accuracy_score, mutual_info_score
```

Ввод [12]:

```python
def get_metrics(classifier, x: pd.DataFrame, y: pd.Series, name: str, training_time: str, num_classes: int):
    """Функция для оценки качества обучения
    Args:
        clasifier объект классификатора с методами predict и predict_proba
        x (pd.DataFrame) признаки
        y (pd.Series) лейблы
        name (str) имя классификатора
        training_time (str) время обучения (включая тюнинг) в формате чч:мм:сс
        num_classes (int) число классов
    Return:
        (dict) структуру строки для датасета с результатами
    """
    preds = classifier.predict(x)
    preds_proba = classifier.predict_proba(x)
    return {
        "model": [name],
        "time": [training_time],
        "accuracy": [accuracy_score(y, preds)],
        "recall": [recall_score(y, preds)],
        "roc_auc": [roc_auc_score(np.eye(num_classes)[y], preds_proba)],
        "f1_score": [f1_score(y, preds)],
        "mutal_score": [mutual_info_score(y, preds)],
    }
```

Обучение baseline

Ввод [13]:

```python
X_train, X_test, y_train, y_test = train_test_split(
    data.drop(columns=['Potability']), data['Potability'], random_state=6875, train_size=0.8
)
```

Ввод [14]:

```python
model = XGBClassifier()
parameters = {
    "n_estimators": [10, 25, 50, 100, 150, 200, 250],
    "max_depth": [5, 10, 15, 20],
    "learning_rate": [1e-2, 1e-3, 1e-5],
    "objective": ["binary:logistic"],
    "eval_metric": ["auc"],
    "tree_method": ["gpu_hist"],
}
```

Ввод [15]:

```python
clf = GridSearchCV(model, parameters, cv=2, verbose=False)
start = time.time()
clf.fit(X_train, y_train)
train_time = time.time() - start
```

Ввод [16]:

```python
metrics_frame = pd.DataFrame(get_metrics(clf.best_estimator_, X_test, y_test, "search", train_time, 2))
metrics_frame
```

Out[16]:

	model	time	accuracy	recall	roc_auc	f1_score	mutal_score
0	search	85.754553	0.682927	0.369919	0.682144	0.466667	0.038577

Ввод [20]:

```python
pickle.dump(clf.best_estimator_, open("../models/baseline.bin", "wb"))
```

На тестовых данных ROC AUC в районе 0.68, что в целом, соответсвует результатам из откртых источников (расшаренный код)

Обогащение данных

Для обогащения данных в случайных местах пропускаем данные и ставим колонку в соотвесвующее состояние. Увеличим датасет в 10 раз (по количеству фичей + исходный)

Ввод [21]:
```
X_train_large = X_train.copy()
y_train_large = y_train.copy()
for col in init_columns:
    if col != 'Potability':
        _temp_frame = X_train.copy()
        _temp_frame[col] = 0.0
        _temp_frame[f'{col}_nan'] = 1
    X_train_large = pd.concat([X_train_large, _temp_frame], ignore_index=True)
    y_train_large = pd.concat([y_train_large, y_train.copy()], ignore_index=True)
print(f"Размерность нового датасета: {len(X_train_large)}")
```

```
Размерность нового датасета: 28820
```

Ввод [22]:
```
clf = GridSearchCV(model, parameters, cv=2, verbose=False)
start = time.time()
clf.fit(X_train_large, y_train_large)
train_time = time.time() - start
```

Ввод [23]:
```
metrics_frame = pd.concat([
    metrics_frame, pd.DataFrame(get_metrics(clf.best_estimator_, X_test, y_test, "large", train_time, 2))
], ignore_index=True)
pickle.dump(clf.best_estimator_, open("../models/robust.bin", "wb"))
```

Ввод [24]:
```
metrics_frame
```

Out[24]:

	model	time	accuracy	recall	roc_auc	f1_score	mutal_score
0	search	85.754553	0.682927	0.369919	0.682144	0.466667	0.038577
1	large	237.992343	0.660061	0.398374	0.672606	0.467780	0.027405

> Как видим качество не сильно упало, однако повысилась устойчивотсь - теперь модель может работать с неполными данными

Сформируем базу данных для Web-api

Ввод [11]:
```
import sqlite3
con = sqlite3.connect("../main_base.db")
```

Ввод [12]:
```
data.to_sql('train_data', con)
```

Out[12]:
```
3276
```

Ввод [ ]:
```

```