Table of contents:


```
import pandas as pd
import numpy as np
import math
import re
import os
import time

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from scipy.stats import spearmanr
from pprint import pprint

from scipy.stats import randint

from itertools import chain
from itertools import product as iterproduct

#!pip install xgboost
from sklearn.impute import SimpleImputer
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler, RobustScaler, StandardScaler, LabelEncoder

#Clustering
#!pip install fuzzy-c-means
from sklearn.cluster import KMeans
#from fcmeans import FCM
from sklearn.cluster import AgglomerativeClustering
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.ensemble import GradientBoostingClassifier

import sklearn.metrics
import plotly.graph_objs as go
from sklearn.metrics import silhouette_score
from sklearn.cluster import DBSCAN

from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline

from sklearn.base import BaseEstimator
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
from xgboost import XGBRegressor
```

```
# Cross-Validation
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, GroupKFold
from sklearn.linear_model import LinearRegression

#GridSearch
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import cross_val_score, cross_validate

from scipy.stats import spearmanr
from scipy import stats
#metrics
from sklearn import metrics
import sklearn.metrics
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

#tensorflow Neural Networks
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.losses import MeanAbsoluteError
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras import regularizers
from keras.models import Model, Sequential
from keras.layers import Embedding, Dense, BatchNormalization, Input, Concatenate, Add
from tensorflow.keras import regularizers, Model
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.utils import plot_model

import pickle

import warnings
warnings.filterwarnings("ignore")

# сохраняю requirements.txt  с текущими версиями библиотек
requirements = !pip freeze
import json
with open('requirements.txt', 'w') as f:
    f.write(json.dumps(requirements))

print('Done')
```

```
Done
```

```
!python --version
```

```
Python 3.8.16
```

```
#Загрузка данных и удаление данных с ошибкой из 'train_updates_20220929.csv' с платформы Kaggle (ниже ссылки для оперативного доступа)

pd.set_option('display.max_columns', 500)
df_train = pd.read_csv('/kaggle/input/novozymes-enzyme-stability-prediction/train.csv', index_col="seq_id")
df_test = pd.read_csv('/kaggle/input/novozymes-enzyme-stability-prediction/test.csv')
df_submission = pd.read_csv('/kaggle/input/novozymes-enzyme-stability-prediction/sample_submission.csv')
df_train_updated = pd.read_csv('../input/novozymes-enzyme-stability-prediction/train_updates_20220929.csv', index_col="seq_id")

all_features_nan = df_train_updated.isnull().all("columns")
drop_indices = df_train_updated[all_features_nan].index
df_train = df_train.drop(index=drop_indices)
swap_ph_tm_indices = df_train_updated[~all_features_nan].index
df_train.loc[swap_ph_tm_indices, ["pH", "tm"]] = df_train_updated.loc[swap_ph_tm_indices, ["pH", "tm"]]
```

```
df_train.shape
```

```
(28981, 4)
```

```
df_train.describe(include='all')
#df_train.set_index('seq_id', inplace=True)
```

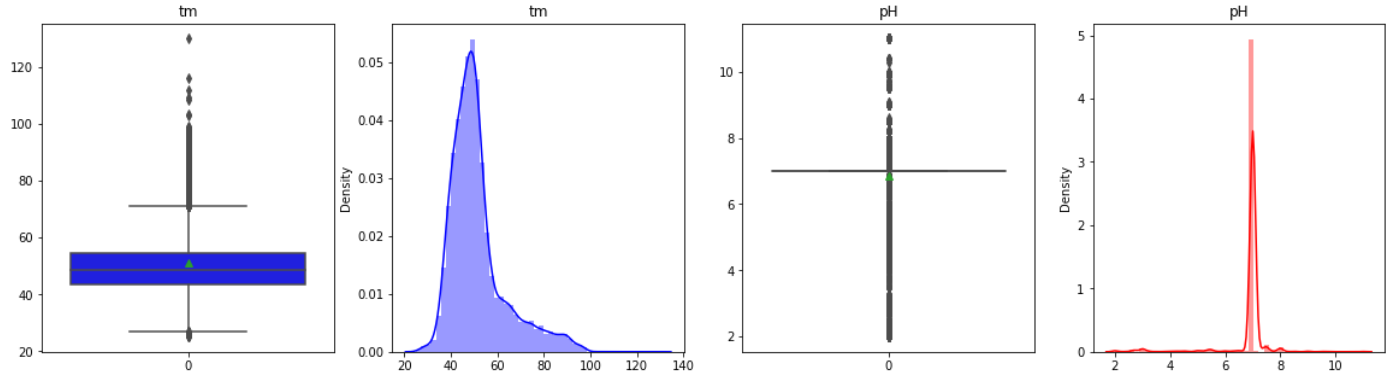
	protein_sequence	pH	data_source	tm
count	28981	28695.000000	28001	28981.000000
unique	27375	NaN	324	NaN
top	MRNTGAGPSPSVSRPPPSAAPLSGAALAAPGDAPSALYAPSALVLT...	NaN	doi.org/10.1038/s41592-020-0801-4	NaN
freq	13	NaN	24525	NaN
mean	NaN	6.872467	NaN	51.360005
std	NaN	0.793184	NaN	12.056717
min	NaN	1.990000	NaN	25.100000
25%	NaN	7.000000	NaN	43.600000

```
df_test.shape
```

```
(2413, 4)
```

```
fig, ax = plt.subplots(ncols=4, figsize=(20, 5))
sns.boxplot(data=df_train.tm, ax=ax[0], color='b', showmeans=True)
sns.distplot(df_train[['tm']], ax=ax[1], color='b', kde=True, bins=50)
ax[0].set_title('tm')
ax[1].set_title('tm')

sns.boxplot(data=df_train.pH, ax=ax[2], color='r', showmeans=True)
sns.distplot(df_train[['pH']], ax=ax[3], color='r', kde=True, bins=50)
ax[2].set_title('pH')
ax[3].set_title('pH')

# we can clearly see some outliers
```

```
Text(0.5, 1.0, 'pH')
```




```
df_test.describe()
```

	seq_id	pH
count	2413.000000	2413.0
mean	32596.000000	8.0
std	696.717422	0.0
min	31390.000000	8.0
25%	31993.000000	8.0
50%	32596.000000	8.0
75%	33199.000000	8.0
max	33802.000000	8.0

Пропуски в данных

```
#пропуски в тестовых данных отутсвуют
df_test.isna().sum()
```

```
seq_id              0
protein_sequence    0
pH                  0
data_source         0
dtype: int64
```

```
df_train.isna().sum()
```

```
protein_sequence      0
pH                  286
data_source         980
tm                    0
dtype: int64
```

- Replace inf values with NaNs
- Drop missing rows with NaN values for pH column, since it is too small and replace missing for data source with 0.

```
# заменим inf значения как np.Nan
df_train['data_source'].replace([np.inf, -np.inf], np.nan, inplace=True)
print(df_train.isna().sum())
```

```
protein_sequence      0
pH                  286
data_source         980
tm                    0
dtype: int64
```

Проверим, есть ли общие источники данных у обоих датасетов

```
df_test.data_source.value_counts()
```

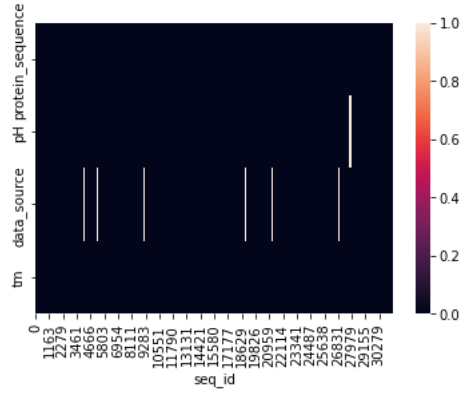
```
Novozymes    2413
Name: data_source, dtype: int64
```

```
df_train[df_train.data_source=='Novozymes']  # данные тренингового датасета взяты из других источников по сравнению с теми, что содержатся в
```

seq_id	protein_sequence	pH	data_source	tm

```
sns.heatmap(df_train.isna().T)
```

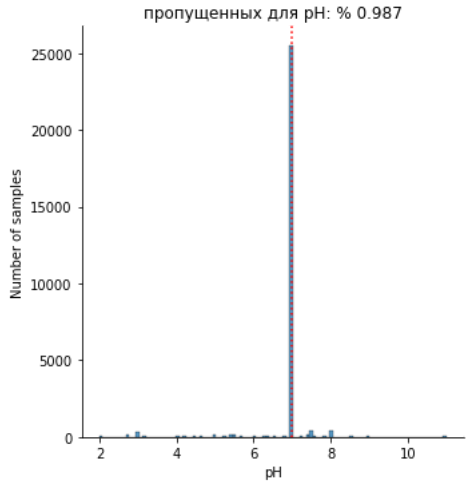
```
<AxesSubplot:xlabel='seq_id'>
```



Явное преобладание кислотного уровня pH (7) в тренировочном датасете

```
bins = len(df_train.pH.value_counts())
p=round((df_train['pH'].isna().sum() / df_train.shape[0]) *100, 3)
sns.displot(df_train.pH, bins=bins)
plt.title('пропущенных для pH: % {}'.format(p))
plt.axvline(7, c='red', linestyle='dotted')
#plt.xlim(0, 10)
```

```
plt.ylabel('Number of samples')
plt.show()
```




```
df_train
```

seq_id	protein_sequence	pH	data_source	tm
0	AAAAKAAALALLGEAPEVVDIWLPAGWRQPFRVFRLERKGDGVLVG...	7.0	doi.org/10.1038/s41592-020-0801-4	75.7
1	AAADGEPLHNEEERAGAGQVGRSLPQESEEQRTGSRPRRRRDLGSR...	7.0	doi.org/10.1038/s41592-020-0801-4	50.5
2	AAAFSTPRATSYRILSSAGSGSTRADAPQVRRLHTTRDLLAKDYYA...	7.0	doi.org/10.1038/s41592-020-0801-4	40.5
3	AAASGLRTAIPAQPLRHLLQPAPRPCLRPFGLLSVRAGSARRSGLL...	7.0	doi.org/10.1038/s41592-020-0801-4	47.2
4	AAATKSGPRRQSQGASVRTFTPFYFLVEPVDTLSVRGSSVILNCSA...	7.0	doi.org/10.1038/s41592-020-0801-4	49.5
...	...	...	...	...
31385	YYMYSGGGSALAAGGGGAGRKGDWNDIDSIKKKDLHHSRGDEKAQG...	7.0	doi.org/10.1038/s41592-020-0801-4	51.8
31386	YYNDQHRLSSYSVETAMFLSWERAIVKPGAMFKKAVIGFNCNVDLI...	7.0	doi.org/10.1038/s41592-020-0801-4	37.2
31387	YYQRTLGAELLYKISFGEMPKSAQDSAENCPSGMQFPDTAIAHANV...	7.0	doi.org/10.1038/s41592-020-0801-4	64.6
31388	YYSFSDNITTVFLSRQAIDDDHSLSLGTISDVVESENGVVAADDAR...	7.0	doi.org/10.1038/s41592-020-0801-4	50.7
31389	YYVPDEYWQSLEVAHKLTFGYGYLTWEWVQGIRSYVYPLLIAGLYK...	7.0	doi.org/10.1038/s41592-020-0801-4	37.6

28981 rows × 4 columns

- Заменим пропущенные значения об источнике данных значением "Unknown"
- Пропущенные значения pH заменим на среднее значение, преобладающее в выборке в целом

```
df_train.pH = df_train.pH.fillna(df_train.pH.mean())
df_train.data_source = df_train.data_source.fillna('Unknown')
```

```
#df_train = df_train.fillna(value=968)
df_train.isna().sum()
```

```
protein_sequence    0
pH                  0
data_source         0
tm                  0
dtype: int64
```

Разведочный анализ данных

Поскольку даннных о о последовательностях мало информативны, создадим несколько дополнительных пременных:

- Длина последовательности аминокислот в протеине
- Уровень pH (щелочной ниже 7 и кислотный -выше 7)

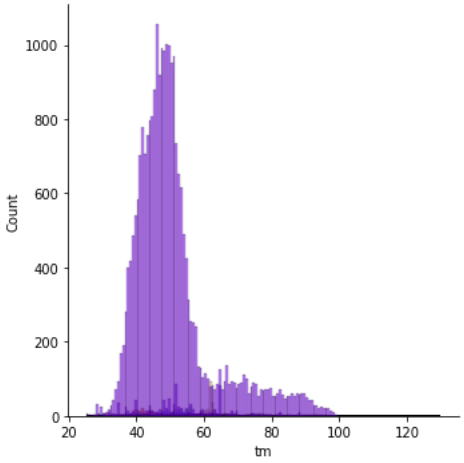
- Данные об источнике данных (категориальный признак) закодированы как численные в соответствии с их рангом (частотой встречаемиости)
- Рассчитано число аминокислот для каждой строки с данными о протеине и рассчитан процент числа каждой аминокислоты к общей длине протеина

Определим, насколько важен параметр источника данных для принятия решения о его дальнейшем удалении либо кодировании числом. Подход с One Hot Encoding представляется нецелесообразным ввиду большого числа источников данных (325 источников)

Для этого рассмотрим распределение целевой переменной термостабильности ('tm') в подвыборках последовательностей аминокислот из разных источников данных для оценки важности этого параметра для дальнейшего анализа

```
sns.displot(df_train, x = 'tm',  hue='data_source', kde_kws={'linewidth': 2}, legend=False, palette='rainbow')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f22c868e3d0>
```



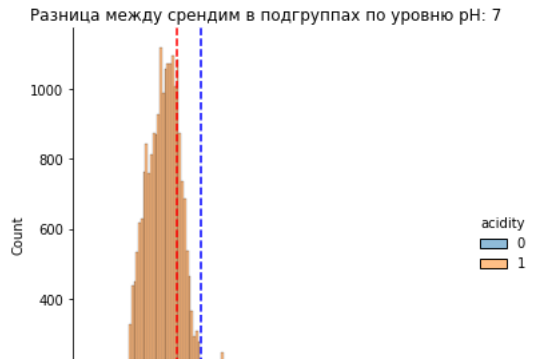
- Можно видеть, что распределение частоты того или иного параметра термостибильности неоднородны в разных группах данных. Поэтому параметр источник данных ('data_soucre') оставлен в датасете.
- Категориальную информацию об источнике удалим.

```
df_train['source'] = df_train['data_source'].rank(pct=True)

del df_train['data_source']

df_train['seq_len'] =[len(s) for s in df_train['protein_sequence']]

df_train['acidity'] = 0
df_train['acidity'] = df_train.apply(lambda row: 0 if row['pH']  > 7 else 1, axis=1)
df_train.sample(1)
```

seq_id	protein_sequence	pH	tm	source	seq_len	acidity
26917	MTEFKAGSAKKGATLFKTRCLQCHTVEKGGPHKVGPNLHGIFGRHS...	5.2	25.2	0.136866	109	1

```
sns.displot(df_train, x = 'tm', hue='acidity')
# нарисуем медианную линию для обоих подгрупп и посчитаем разницу средних

mean_acid = round(df_train[df_train.acidity==1].tm.mean())
mean_base = round(df_train[df_train.acidity==0].tm.mean())
ph =  mean_base - mean_acid
plt.axvline(df_train[df_train.acidity==0].tm.mean(), color='blue', ls='--')
plt.axvline(df_train[df_train.acidity==1].tm.mean(), color='red', ls='--')
plt.title('Разница между срендим в подгруппах по уровню pH: {}'.format(ph))
```

```
Text(0.5, 1.0, 'Разница между срендим в подгруппах по уровню pH: 7')
```




```
aminoacids = ['A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']
len(aminoacids)

# создаем колонки с признаками пропорции той или иной аминокислоты в общем в последовательности
for amino in aminoacids:
    df_train[amino] = (df_train.protein_sequence.str.count(amino, re.I) / df_train.seq_len) *100
df_train.sample()
```

seq_id	protein_sequence	pH	tm	source	seq_len	acidity	A	R	N	
21725	MRFETLQLHAGYEPEPTTLSRQVPIYPTTSYVFKSPEHAANLFALK...	7.0	88.2	0.576895	420	1	11.428571	5.47619	2.857143	2.619

Также отдельно создадим копию датасета, где создадим колонки с признаками об общем числе аминокислот для проверки идеи о том, что количество аминокислот, а не только их пропорция имеет значение для целевого показателя

Затем сохраним полученные данные в отдельный датасет.

```
df_train_amino_count = df_train.copy()
for amino_count in aminoacids:
    df_train_amino_count[amino_count] = df_train_amino_count.protein_sequence.str.count(amino_count, re.I)
df_train_amino_count.reset_index(inplace=True)
df_train_amino_count.sample()
```

	seq_id	protein_sequence	pH	tm	source	seq_len	acidity	A	R	N	D	C	E	Q	
3867	3992	MAEDTDGGIRFNSLCFINDHVGFQGSIKTSPSDFIVIEIDEQGQLV...	7.0	50.9	0.576895	701	1	44	42	34	38	14	58	25	4

```
df_train_amino_count= df_train_amino_count[['seq_id', 'protein_sequence', 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K',
       'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']]
df_train_amino_count.columns = [str(col) + '_count' for col in df_train_amino_count.columns]
df_train_amino_count.sample()
```

	seq_id_count	protein_sequence_count	A_count	R_count	N_count	D_count	C_count	E_count	Q_c
14551	15636	MLKEHSKKQHLRREVQEKLKNEAIVAAQTLSTAVVDHLNAKVAQAY...	16	6	11	4	0	13	

Переименуем названия колонок, чтобы затем присоединить к итоговому датафрейму

```
df_train_amino_count.rename(columns={'seq_id_count':'seq_id', 'protein_sequence_count': 'protein_sequence'}, inplace=True)

df_train_amino_count.sample()
```

	seq_id	protein_sequence	A_count	R_count	N_count	D_count	C_count	E_count	Q_count	
23356	25579	MSSFTKDEFDCHILDEGFTAKDILDQKINEVSSSDDKDAFYVADLG...	40	19	15	35	12	27	16	

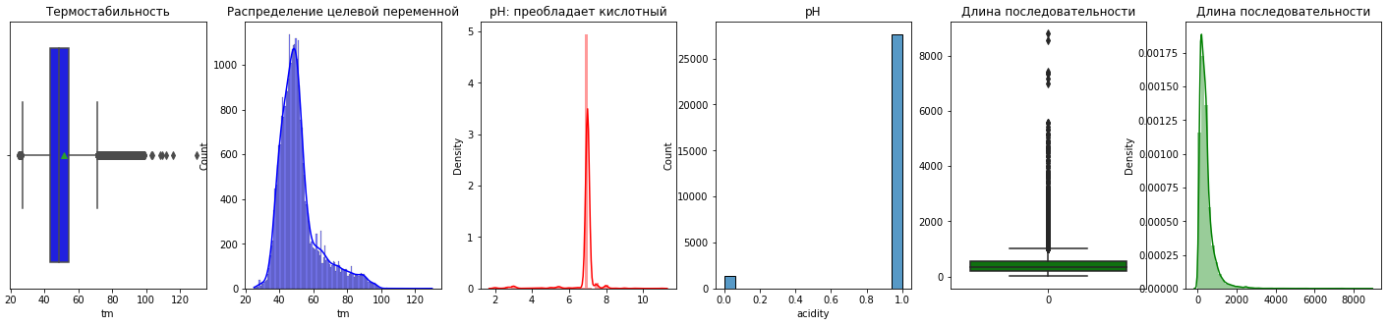
```
# список подсчитанного числа аминокислот в последовательностях
aminoacids_count = ['A_count', 'R_count', 'N_count',
       'D_count', 'C_count', 'E_count', 'Q_count', 'G_count', 'H_count',
```

```
       'I_count', 'L_count', 'K_count', 'M_count', 'F_count', 'P_count',
       'S_count', 'T_count', 'W_count', 'Y_count', 'V_count']

df_train = pd.merge(df_train, df_train_amino_count, how='inner', on =['protein_sequence', 'seq_id'])
df_train.sample()
```

	protein_sequence	seq_id	pH	tm	source	seq_len	acidity	A	R	N	
305	ALPTPRHLHTSPWRADCSRASLTRLRRQAYARLYPVLLVKQDGSTI...	319	7.0	50.0	0.576895	109	1	8.256881	16.513761	0.0	5.5

Оценка распределения числовых переменных с помощью гистрограмм

```
fig, ax = plt.subplots(ncols=6, figsize=(25, 5))
sns.boxplot(df_train.tm, ax=ax[0], color='b', showmeans=True)
sns.histplot(df_train.tm, ax=ax[1], color='b', kde=True)
ax[0].set_title('Термостабильность')
ax[1].set_title('Распределение целевой переменной')
ax[2].set_title('pH: преобладает кислотный')
sns.distplot(df_train[['pH']], ax=ax[2], color='r', kde=True, bins=50)
sns.histplot(data=df_train.acidity, ax=ax[3])
ax[3].set_title('pH')
ax[4].set_title('Кислотный vs Щелочной pH')

sns.boxplot(data=df_train.seq_len, ax=ax[4], color='g')
sns.distplot(df_train[['seq_len']], ax=ax[5], color='g', kde=True, bins=50)
ax[4].set_title('Длина последовательности')
ax[5].set_title('Длина последовательности')
plt.savefig('pic1.png')
```



Для остальных параметров (процентное соотошение аминокислот) обратимся к таблице ниже

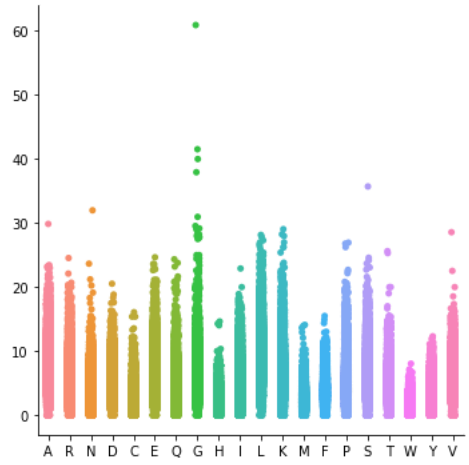
```
df_train.describe()
```

	seq_id	pH	tm	source	seq_len	acidity	A	R	N	
count	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000	28981.0
mean	15733.571029	6.872467	51.360005	0.500017	450.468617	0.953659	7.848316	5.427759	4.253150	5.5
std	9256.053131	0.789261	12.056717	0.181189	415.159049	0.210226	2.782231	2.239725	1.861235	1.7
min	0.000000	1.990000	25.100000	0.000086	5.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	7511.000000	7.000000	43.600000	0.576895	212.000000	1.000000	5.905512	3.971631	3.018868	4.4
50%	15575.000000	7.000000	48.800000	0.576895	351.000000	1.000000	7.544910	5.196182	4.081633	5.4
75%	23896.000000	7.000000	54.600000	0.576895	537.000000	1.000000	9.388458	6.617647	5.263158	6.5
max	31389.000000	11.000000	130.000000	0.576895	8798.000000	1.000000	29.865772	24.539877	31.989924	20.5

Оценим преобладание той или ной аминокислоты в последовательностях аминокислот в целом на графике ниже

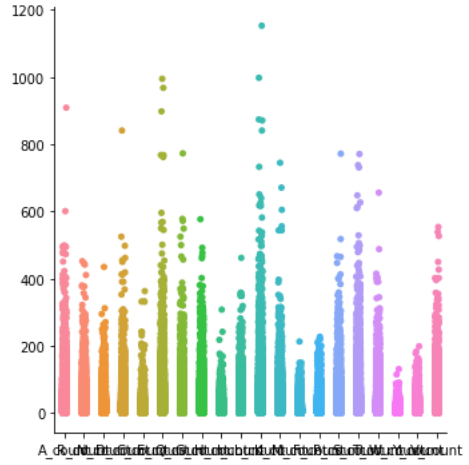
```
sns.catplot(data=df_train[aminoacids])
```

```
<seaborn.axisgrid.FacetGrid at 0x7f22c17bf990>
```




```
# График отдельно для количества аминокислот в белках
sns.catplot(data=df_train[aminoacids_count])
```

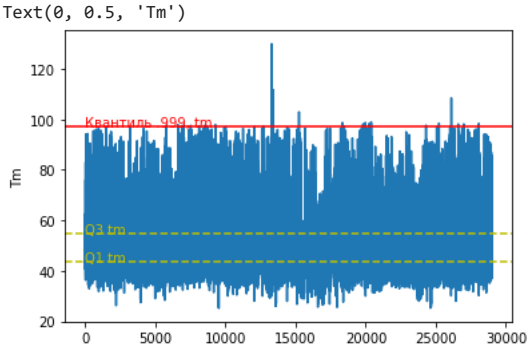
```
<seaborn.axisgrid.FacetGrid at 0x7f2280b1e0d0>
```



Выбросы

```
low_tm, high_tm = np.percentile(df_train.tm, [25,  75])
line = df_train.tm.quantile(.999)  # за .999 квантилем есть выбросы (знчение, которое случайная величина не превысит с вероятностью .999)
df_train.tm.plot()
plt.axhline(y=line, c='r')
plt.text(0.1, line, 'Квантиль .999, tm', c='r')
plt.axhline(y= low_tm,  c='y', linestyle='--')
plt.axhline(y= high_tm,  c='y', linestyle='--')
plt.text(0.1, high_tm, 'Q3 tm', c='y')
plt.text(0.1, low_tm, 'Q1 tm', c='y')
plt.ylabel('Tm')
```

```
Text(0, 0.5, 'Tm')
```



```
#q_low_tm = df_train_train_train['tm'].quantile(0.01)
q_low_ph = df_train['pH'].quantile(0.05)
q_hi_ph  = df_train['pH'].quantile(0.95)
outliers_pH = df_train[(df_train['pH'] > q_hi_ph) | (df_train['pH'] < q_low_ph)]
outliers_pH.shape
```

Посмотрим на распределение целевой перемменной в группе выбросов и в общем по выборке

```
sns.distplot(outliers_pH.tm, hist=True, kde=True, bins=50, kde_kws={'linewidth': 2}, color='red')
sns.distplot(df_train[df_train.pH <q_hi_ph].tm, hist=True, kde=True, bins=50, color = 'blue')
```

```
#df_train = df_train[df_train.pH <q_hi_ph]
low_ph, high_ph = np.percentile(df_train.pH, [25,  75])
line=max(df_train[df_train.pH<q_hi_ph].pH)

df_train.pH.plot()
plt.axhline(y=line, c='r')
plt.text(0.1, line, 'Max pH < 0.99 quantile', c='r')
plt.axhline(y= low_ph,  c='y', linestyle='--')
plt.axhline(y= high_ph,  c='y', linestyle='--')
plt.text(0.1, high_ph, 'Q3 pH', c='y')
plt.text(0.1, low_ph, 'Q1 pH', c='y')
plt.ylabel('pH')
```

```
df_train.to_csv('df_train_before_scaling.csv')
```

Теперь перед проведеним шкалирования подготовим тестовые данные

Преобразование тестового датасета: создание новых переменных

Для предсказания значений теомостабильности на тестовом датасете требуется приести его к тому же формату, что и тренирочный датасет, то есть создать дополнительные переменные, такие как длина последовательности, уровень pH, количество аминокислот в последовательности белка и его пропорция в процентах.

Повторим манипуляции и добавим те же переменные, что ранее добавили в тренировочный датасет:

- длину последовательности аминокислот
- процентное соотношение аминокислот в белке
- количество аминокислот в последовательности

```
df_test = pd.read_csv('/kaggle/input/novozymes-enzyme-stability-prediction/test.csv')
#df_test.rename(columns={'seq_id': 'id'}, inplace=True)
df_test.set_index('seq_id',inplace=True)
df_test.sample()
```

```
df_test['seq_len'] =[len(s) for s in df_test['protein_sequence']]
df_test['source'] = max(df_train.source)* 1.1  # заменим на число немного большее, чем максимальный ранк в тренировочном датасете
del df_test['data_source']
df_test['seq_len'] =[len(s) for s in df_test['protein_sequence']]
df_test['acidity'] = 0
df_test['acidity'] = df_test.apply(lambda row: 0 if row['pH']  > 7 else 1, axis=1)
# создаем колонки с признаками пропорции той или иной аминокислоты в общем в последовательности

for amino in aminoacids:
    df_test[amino] = (df_test.protein_sequence.str.count(amino, re.I) / df_test.seq_len) *100
df_test.sample()
```

```
max(df_train.source)
df_test_amino_count = df_test.copy()
for amino_count in aminoacids:
    df_test_amino_count[amino_count] = df_test_amino_count.protein_sequence.str.count(amino_count, re.I)
df_test_amino_count.reset_index(inplace=True)
df_test_amino_count= df_test_amino_count[['seq_id', 'protein_sequence', 'A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K',
       'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V']]
df_test_amino_count.columns = [str(col) + '_count' for col in df_test_amino_count.columns]
df_test_amino_count.rename(columns={'seq_id_count':'seq_id', 'protein_sequence_count': 'protein_sequence'}, inplace=True)
df_test = pd.merge(df_test, df_test_amino_count, how='inner', on =['protein_sequence', 'seq_id'])
```

```
df_test.set_index('seq_id', inplace=True)
df_test.reset_index(inplace=True)
df_test.sample() # 31,390 первый индекс в тестовом датасете
```

Объединима датасеты для того, чтобы провести стандартизацию и нормализацию для числовых переменных. ДЛя этого тренинировочный датасет возьмем без целевой переменной 'tm' и без колонки 'protein sequence', затем после стандартизации присоединим их обратно

```
# создадим датасет с данными 'tm' и 'protein sequence'
df_tm = df_train[['seq_id', 'protein_sequence', 'tm']]


train_indexes = df_train.seq_id.tolist()
train_indexes[-1]


test_indexes = df_test.reset_index().seq_id.tolist()
test_indexes[0]


merge_columns= ['protein_sequence', 'seq_id', 'pH', 'source', 'seq_len', 'acidity', 'A',
       'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P',
       'S', 'T', 'W', 'Y', 'V', 'A_count', 'R_count', 'N_count', 'D_count',
       'C_count', 'E_count', 'Q_count', 'G_count', 'H_count', 'I_count',
       'L_count', 'K_count', 'M_count', 'F_count', 'P_count', 'S_count',
       'T_count', 'W_count', 'Y_count', 'V_count']


df_train_to_merge = df_train[merge_columns]
df_train_to_merge.set_index('seq_id', inplace=True)
df_train_to_merge.sample()


df_test.reset_index(inplace=True)


df_train_to_merge.reset_index(inplace=True)
df_train_to_merge.sample()


df_final = pd.concat([df_train_to_merge, df_test])


df_final.sample()


df_final.shape


# оценим распределение данных в объединенном датасете до масштабирования
df_final.describe()
```

2 Практическая часть

Комментарий:

Далее чтобы провести кластеризацию и сохранить информацию, содержащуюся в тестовом датасете (в нем нет меток, но есть данные по протеинам с щелочным pH, которых менее 1% в тренировочной части), будем далее работать на объединенном датасете (для вычисления эмбеддингом и нахождения кластеров), затем снова разделим на две части по индексам, сохраненным выше (для предсказания термостабильности)

Эмбддинги

2.1 - Эмбеддинги на объединенных датасетах

```
# Source: https://github.com/cran2367/sgt/tree/master/python/sgt-package#sgt-class-def
!pip install sgt
import sgt
sgt.__version__
from sgt import SGT
```

SGT: SGT embedding embeds the long- and short- term patterns in a sequence into a finite-dimensional vector. The advantage of SGT embedding is that we can easily tune the amount of long- / short- term patterns without increasing the computation.

kappa - Tuning parameter, kappa > 0, to change the extraction of long-term dependency. Higher the value the lesser the long-term dependency captured in the embedding. Typical values for kappa are 1, 5, 10.

lengthsensitive - If set to false then the embedding of two sequences with similar pattern but different lengths will be the same.

```
df_final.sample()
```

```
corpus = df_final.copy()
corpus['protein_sequence'] = corpus['protein_sequence'].map(list)
#corpus.set_index('id', inplace=True)
corpus = corpus [['seq_id', 'protein_sequence']]
# change index column name becuase otherwise SGT does not work
corpus.columns = ['id', 'sequence']
corpus.sample(1)
```

```
%%time
# Compute SGT embeddings
sgt = SGT(kappa=1,
          lengthsensitive=False,
          mode='multiprocessing')
embeddings = sgt.fit_transform(corpus)
embeddings.to_csv('embeddings.csv')
embeddings.sample()

embeddings['id'] = embeddings['id'].astype('int32')
embeddings.sample()
```

```
embeddings = pd.read_csv('/kaggle/input/embeddings/embeddings.csv')
del embeddings['Unnamed: 0']
embeddings.sample()
```

	id	('A', 'A')	('A', 'C')	('A', 'D')	('A', 'E')	('A', 'F')	('A', 'G')	('A', 'H')	('A', 'I')	('A', 'K')	('A', 'L')	('A', 'M')	('A', 'N')	('A', 'P')	(
30245	32654.0	0.002752	0.00165	0.004839	0.001531	0.004618	0.004929	0.0	0.012063	0.00291	0.020345	0.0	0.004382	0.010687	0.006

```
embeddings.shape
```

```
(31394, 401)
```

```
# сохраним названия колонок эмбеддингов, чтобы затем исключить эти колонки с данными при необходимости
emb_cols = embeddings.columns.tolist()
emb_cols.remove('id')
```

2.2 - Масштабирование данных

Для целей применения алгоритмов, которые основываются на измерении расстояний, например, k-ближайщих соседей, проведем нормализацию/стандартизацию переменных в датасете с эмбеддингами и объединенном датасете и затем объединим их для проведения кластеризации

```
df_final.sample()
```

```
mm = MinMaxScaler()
ss = StandardScaler()
embeddings[emb_cols] = mm.fit_transform(embeddings[emb_cols])
embeddings[emb_cols] = ss.fit_transform(embeddings[emb_cols])
embeddings.sample()
```

```
columns_to_scale = ['pH', 'source', 'seq_len', 'acidity', 'A',
       'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I', 'L', 'K', 'M', 'F', 'P',
       'S', 'T', 'W', 'Y', 'V', 'A_count', 'R_count', 'N_count', 'D_count',
       'C_count', 'E_count', 'Q_count', 'G_count', 'H_count', 'I_count',
       'L_count', 'K_count', 'M_count', 'F_count', 'P_count', 'S_count',
       'T_count', 'W_count', 'Y_count', 'V_count']
```

```
df_final[columns_to_scale] = ss.fit_transform(df_final[columns_to_scale])
df_final.sample()
```

```
df_final[columns_to_scale] = mm.fit_transform(df_final[columns_to_scale])
df_final.sample()
```

```
# последний раз взглянем на описание переменных после масштабирования
df_final.describe()
```

Теперь построим финальные графики распределения переменных для всего датасета в целом. Но для целевой переменной 'tm' график построим на основе данных только тренировочного датасета, т.к. в тестовом ее требуется предсказать

```
fig, ax = plt.subplots(ncols=7, figsize=(25, 5))
sns.boxplot(df_train.tm, ax=ax[0], color='b', showmeans=True)
sns.histplot(df_train.tm, ax=ax[1], color='b', kde=True)
ax[0].set_title('Термостабильность')
ax[1].set_title('Термостибильность')

sns.distplot(df_final[['pH']], ax=ax[2], color='r', kde=True, bins=50)
ax[2].set_title('pH: в выборке в целом')
sns.distplot(df_train['pH'], ax=ax[3], color='r', kde=True, bins=50)
ax[3].set_title('pH: только в тренировочной выборке')
sns.histplot(data=df_train.acidity, ax=ax[4], color='r')
ax[4].set_title('pH: только в тренировочной выборке')

sns.distplot(df_train[['seq_len']], ax=ax[5], color='g', kde=True, bins=50)
ax[5].set_title('Длина последовательности')
sns.boxplot(df_train['seq_len'], ax=ax[6], color='g')
ax[6].set_title('Длина последовательности')
plt.savefig('pic1.png')
```

```
corr, p = spearmanr(df_train.tm, df_train.pH)
print(corr, round(p, 3))
corr, p = spearmanr(df_train.tm, df_train.A)
print(corr, round(p, 3))
corr, p = spearmanr(df_train.tm, df_train.C_count)
print(corr, round(p, 3))
corr, p = spearmanr(df_train.tm, df_train.S_count)
print(corr, round(p, 3))
corr, p = spearmanr(df_train.tm, df_train.K_count)
print(corr, round(p, 3))
```

```
sns.pairplot(df_train, corner = True)
plt.savefig('fig.png')
# there is a potential correlation between those variables
```

```
plt.figure(figsize=(20, 20))
corr = df_final.corr(method='spearman')
mask = np.triu(np.ones_like(corr, dtype=bool))
filtered_df = corr[((corr >= .1) | (corr <= -.1)) & (corr !=1)]
sns.heatmap(filtered_df, cmap='vlag', mask=mask, annot=True)
plt.savefig('corr_fig_scaled.png')
```

```
df_train.to_csv('final_scaled_train_df.csv')
df_test.to_csv('final_scaled_test_df.csv')
```

```
df_final.rename(columns={'seq_id': 'id'}, inplace=True)


# объединяем с эмбеддингами
df_all = embeddings.merge(df_final, how='inner', on='id')


del df_all['protein_sequence']
df_all.set_index('id', inplace=True)
df_all.to_csv('df_train_and_test_scaled_embeddings.csv')
```

2.3 - Снижение размерности и кластеризация

Загрузим сохраненный на предыдущем этапе объединенный датасет с эмбеддингами для терировочной и тестовой части без меток тренировочной части

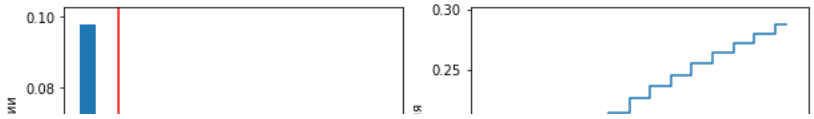
Далее для проверки эффективовсти методов кластеризации будем использовать как датасет только с эмбеддингами и индексами, так и итоговый датасет, объединяющий матрицу эмбеддингов и финальную матрицу со всеми масштабированными параметрами (длина последовательности, pH).

```
df_all = pd.read_csv('/kaggle/input/working/df_train_and_test_scaled_embeddings.csv')


del df_all['protein_sequence']


del df_all['Unnamed: 0']
df_all.set_index('id', inplace=True)
```

проверим ценность использования метода снижения размерности PCA на общей матрице со всеми данныи, включая эмбеддинги

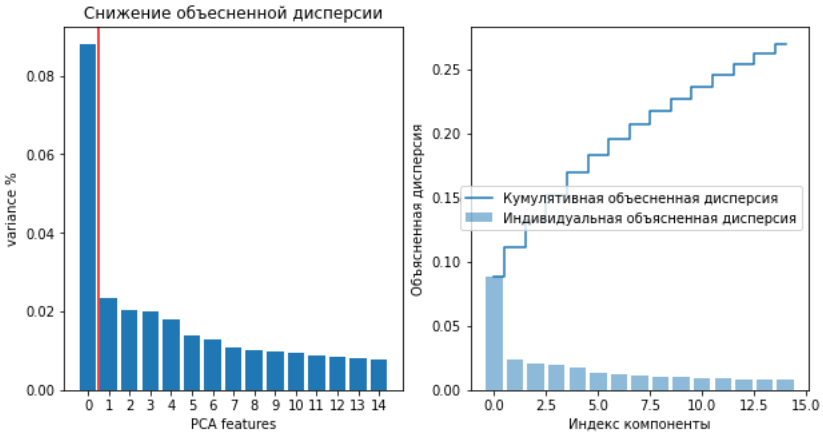
```
n=15
pca = PCA(n_components=n)

X = pca.fit_transform(df_all)
print(np.sum(pca.explained_variance_ratio_))
print(np.sum(pca.explained_variance_))
df_pca = pd.DataFrame(data=X)

features = range(pca.n_components_)
# кумулятивная сумма сокственных векторов, поможет визуализировать объясняемую компонентами дисперсию
exp_var_pca = pca.explained_variance_ratio_
cumsum = np.cumsum(exp_var_pca)
explained_variance = pca.explained_variance_

f, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].bar(features, pca.explained_variance_ratio_)
ax[0].set_xlabel('PCA компоненты')
ax[0].set_ylabel('% объясненной дисперсии')
ax[0].set_xticks(features)
ax[0].axvline(x=1.5, c='r')

ax[1].bar(range(0,len(exp_var_pca)), exp_var_pca, alpha=0.5, align='center', label='Индивидуальная объясненная дисперсия')
ax[1].step(range(0,len(cumsum)), cumsum, where='mid',label='Кумулятивная объясненная дисперсия')
ax[1].set_ylabel('Обьясненная дисперсия')
ax[1].set_xlabel('Индекс компоненты')
ax[1].legend(loc='best')
```

```
0.2873974529924973
127.60853387141808
<matplotlib.legend.Legend at 0x7f23102686d0>
```




```
#проверим PCA отдельно только на данных матрицы эмбеддингов
n=15
pca = PCA(n_components=n)

X = pca.fit_transform(df_all[emb_cols])
print(np.sum(pca.explained_variance_ratio_))
print(np.sum(pca.explained_variance_))
df_pca = pd.DataFrame(data=X)

features = range(pca.n_components_)
#
exp_var_pca = pca.explained_variance_ratio_
cum_sum = np.cumsum(exp_var_pca)
explained_variance = pca.explained_variance_

#
# Cumulative sum of eigenvalues; This will be used to create step plot
# for visualizing the variance explained by each principal component.

f, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].bar(features, pca.explained_variance_ratio_)
ax[0].set_xlabel('PCA features')
ax[0].set_ylabel('variance %')
ax[0].set_xticks(features)
ax[0].axvline(x=0.5, c='r')
ax[0].set_title('Снижение объесненной дисперсии')
#
# Create the visualization plot

ax[1].bar(range(0,len(exp_var_pca)), exp_var_pca, alpha=0.5, align='center', label='Индивидуальная объясненная дисперсия')
ax[1].step(range(0,len(cum_sum)), cum_sum, where='mid',label='Кумулятивная объесненная дисперсия')
ax[1].set_ylabel('Объясненная дисперсия')
ax[1].set_xlabel('Индекс компоненты')
ax[1].legend(loc='best')
```

```
0.26952442347175454
107.81320358643342
<matplotlib.legend.Legend at 0x7f2307bac210>
```

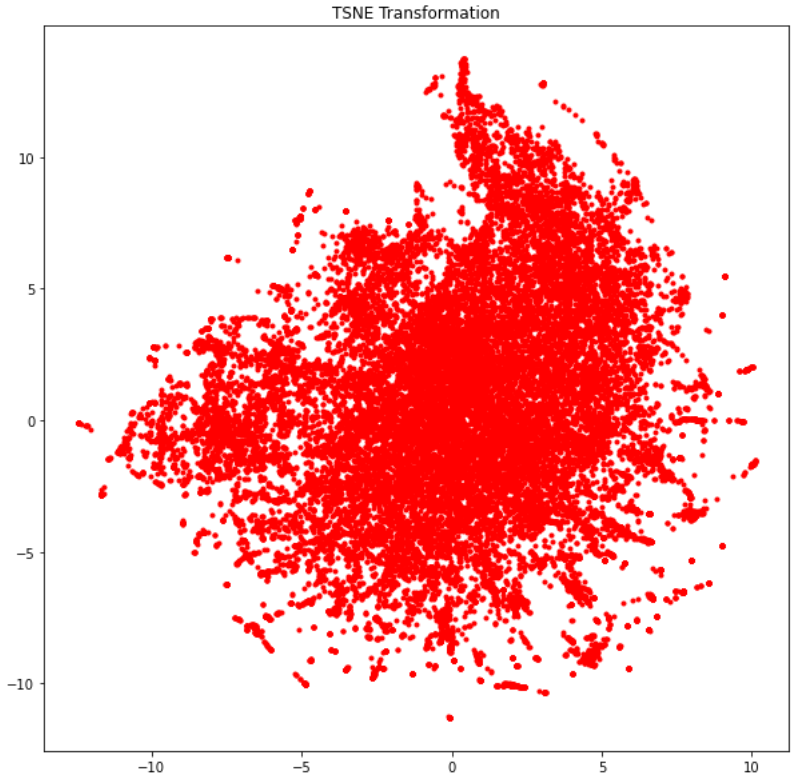


Подитог: метод PCA показывает небольшую эффективность даже при 10 главных компонентах, в связи с чем целесообразность его использования не обоснована. Кластеризация исходной матрицы показала большую ошибку. Вычислим инетрцию метода k-ближийщих соседей на всем датасете

```
k=list(range(2,15))

ssd=[]

for i in k:
    kmeans=KMeans(n_clusters=i).fit(df_all)
```

```
    ssd.append(kmeans.inertia_)

plt.plot(k, ssd, 'o-')
plt.xlabel('k-ближайщих соседей на общем датасете')
#plt.ylabel('Sum of squared error')
ii=kmeans.inertia_
plt.title('Растояние от точек до центроидов составляет:{}'.format(ii))
plt.show()
```

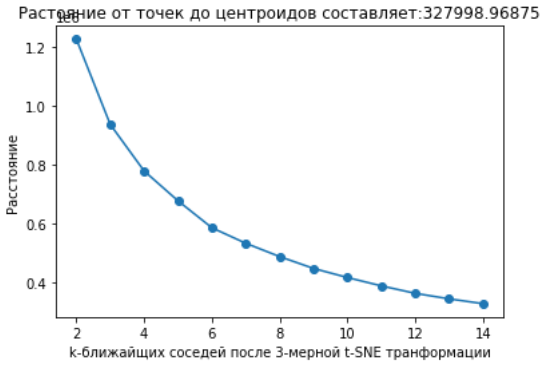

```
# Трансформируем данные с помощью метода t-sne с тремя компонентами

from sklearn.manifold import TSNE
tsne = TSNE(n_components=3, n_iter=400)
tsne_3 = tsne.fit_transform(df_all)


#Визуализируем полученные путем t-SNE Транформации компоненты
plt.figure(figsize=(10,10))
plt.plot(tsne_3[:,0], tsne_3[:,1], 'r.')
plt.title('TSNE Transformation')
plt.show()
plt.savefig('creature.png')
```




```
<Figure size 432x288 with 0 Axes>
```

```
k=list(range(2,15))

ssd=[]

for i in k:
```

```
    kmeans=KMeans(n_clusters=i).fit(tsne_3)
    ssd.append(kmeans.inertia_)

plt.plot(k, ssd, 'o-')
plt.xlabel('k-ближайших соседей после 3-мерной t-SNE транформации')
plt.ylabel('Расстояние')
ii=kmeans.inertia_
plt.title('Растояние от точек до центроидов составляет:{}'.format(ii))
plt.show()
```



k-ближайших соседей

Построим несколько графиков для визуализации 10, 12 и 15 кластеров (оптимальное число, согласно "методу локтя"), полученных на основе исходной матрицы следующими методами

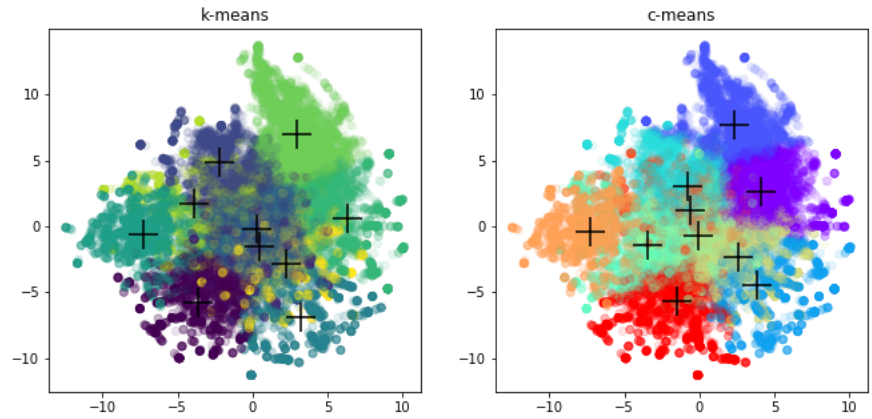
- k-means
- c-means
- Agglomerative Clustering

Также отдельно построим 3D модель для компонент, полученных путем трансформации t-SNE.

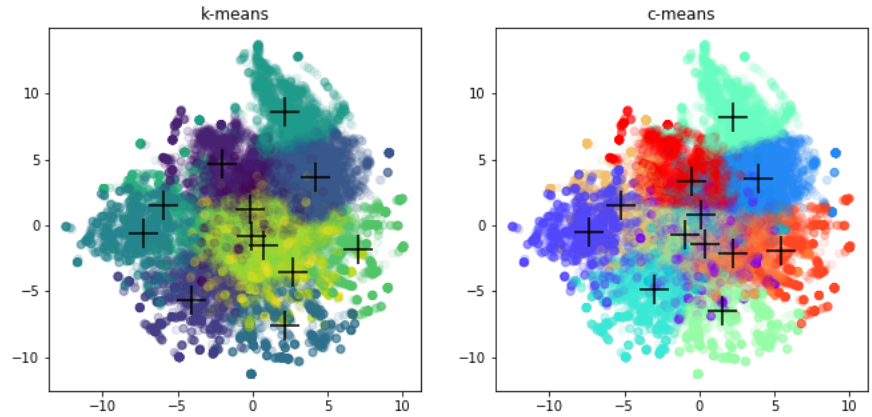
- 3D модель DBSCAN

```
for i in  [10, 12, 15]:
    kmeans = KMeans(n_clusters=i, max_iter=500)
    kmeans.fit(tsne_3)
    centroids = kmeans.cluster_centers_
    km_labels = kmeans.predict(tsne_3)

    fcm=FCM(n_clusters=i)
    fcm.fit(tsne_3)
    fcm_centers=fcm.centers
    fcm_labels=fcm.predict(tsne_3)

    print(kmeans.inertia_)
    f, axes = plt.subplots(1, 2, figsize=(11, 5))
    axes[0].scatter(tsne_3[:, 0], tsne_3[:, 1], c = km_labels, alpha=.1, cmap='viridis')
    axes[0].scatter(centroids[:, 0], centroids[:, 1], marker="+", s=500, c='black')
    axes[0].set_title('k-means')
    axes[1].scatter(tsne_3[:, 0], tsne_3[:, 1], c = fcm_labels, alpha=.1, cmap='rainbow')
    axes[1].scatter(fcm_centers[:, 0], fcm_centers[:, 1], marker="+", s=500, c='black')
    axes[1].set_title('c-means')
    plt.show()
```

```
416586.46875
```




```
364985.5
```



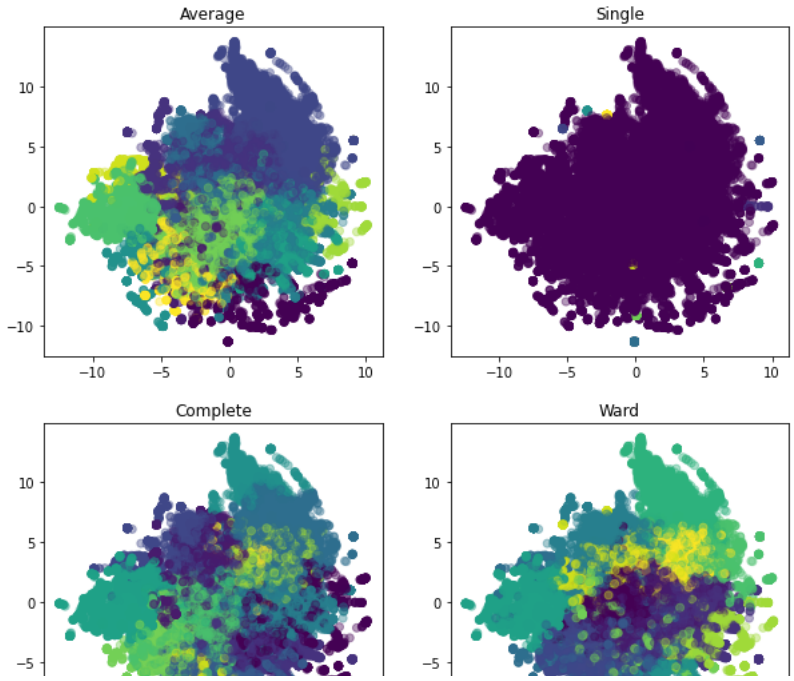

```
312319.1875
```


```
round(fcm.error, 3)
```

Агломеративная кластеризация


```
from sklearn.metrics import pairwise_distances
AG1 = AgglomerativeClustering(n_clusters=15, linkage='average').fit(tsne_3)
print(AG1.n_features_in_)

AG2 = AgglomerativeClustering(n_clusters=15, linkage='complete').fit(tsne_3)
print(AG2.n_features_in_)

AG3 = AgglomerativeClustering(n_clusters=15, linkage='single').fit(tsne_3)  # использует мин. расстояния между наблюдениями в двух кластерах
print(AG3.n_features_in_)

AG4 = AgglomerativeClustering(n_clusters=15, linkage='ward').fit(tsne_3)
print(AG4.n_features_in_)

ag4_labels = AG4.labels_

f, ax = plt.subplots(2,2, figsize=(10,10))
ax[0, 0].set_title("Average")
ax[1, 0].set_title("Complete")
ax[0, 1].set_title("Single")
ax[1, 1].set_title("Ward")
ax[0, 0].scatter(tsne_3[:, 0], tsne_3[:, 1], c=AG1.labels_, alpha =.4, cmap='viridis')
ax[1, 0].scatter(tsne_3[:, 0], tsne_3[:, 1], c=AG2.labels_, alpha =.4, cmap='viridis')
ax[0, 1].scatter(tsne_3[:, 0], tsne_3[:, 1], c=AG3.labels_, alpha =.4, cmap='viridis')
ax[1, 1].scatter(tsne_3[:, 0], tsne_3[:, 1], c=AG4.labels_, alpha =.4, cmap='viridis')
plt.show()
```

```
3
3
3
3
```



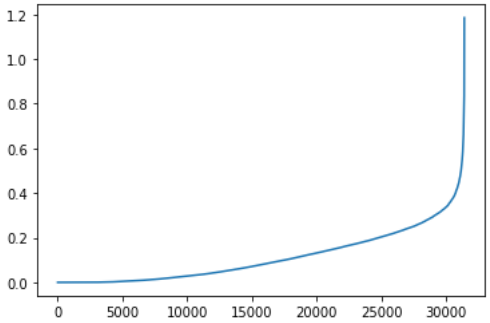
DBSCAN и 3D визуализация

Оптимальное значение для эпсилона для дальнейшей работы алгоритма DBSCAN будет найдено в точке максимальной кривизны

Источник кода для вычислений: https://towardsdatascience.com/explaining-dbscan-clustering-18eaf5c83b31

```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(tsne_3)
distances, indices = nbrs.kneighbors(tsne_3)
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
```

```
[<matplotlib.lines.Line2D at 0x7f22fc13a4d0>]
```



```
#Далее также циклом подберем наиболее оптимальные значения эпсилон

eps = [.2, .22, .23, .25, .2, .22, .23, .25, .2, .22, .23, .25]
mins= [2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4]
for i, j in zip(eps, mins):
    db = DBSCAN(eps=i, min_samples = j).fit(tsne_3)
    db_labels = db.labels_
    # Number of clusters in labels, ignoring noise if present.
    n_clusters_ = len(set(db_labels)) - (1 if -1 in db_labels else 0)
    n_noise_ = list(db_labels).count(-1)
    print(i, j)
    print('Estimated number of clusters: %d' % n_clusters_)
    print('Estimated number of noise points: %d' % n_noise_)
```

```
    print("Silhouette Coefficient: %0.3f" % sklearn.metrics.silhouette_score(tsne_3, db_labels))
    print('')
```

```
Estimated number of noise points: 6594
Silhouette Coefficient: 0.298

0.22 2
Estimated number of clusters: 4055
Estimated number of noise points: 5445
Silhouette Coefficient: 0.332

0.23 2
Estimated number of clusters: 3998
Estimated number of noise points: 4931
Silhouette Coefficient: 0.343

0.25 2
Estimated number of clusters: 3772
Estimated number of noise points: 3951
Silhouette Coefficient: 0.357

0.2 3
Estimated number of clusters: 2255
Estimated number of noise points: 10368
Silhouette Coefficient: 0.151

0.22 3
Estimated number of clusters: 2343
Estimated number of noise points: 8869
Silhouette Coefficient: 0.202

0.23 3
Estimated number of clusters: 2374
Estimated number of noise points: 8179
Silhouette Coefficient: 0.223

0.25 3
Estimated number of clusters: 2334
Estimated number of noise points: 6827
Silhouette Coefficient: 0.257

0.2 4
Estimated number of clusters: 1372
Estimated number of noise points: 13552
Silhouette Coefficient: 0.027

0.22 4
Estimated number of clusters: 1494
Estimated number of noise points: 12093
Silhouette Coefficient: 0.084

0.23 4
Estimated number of clusters: 1543
Estimated number of noise points: 11414
Silhouette Coefficient: 0.106

0.25 4
Estimated number of clusters: 1613
Estimated number of noise points: 9890
Silhouette Coefficient: 0.158
```

```
db = DBSCAN(eps=.25, min_samples = 2).fit(tsne_3)
db_labels = db.labels_


print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Silhouette Coefficient: %0.3f" % sklearn.metrics.silhouette_score(tsne_3, db_labels))
```

```
Estimated number of clusters: 1613
Estimated number of noise points: 9890
Silhouette Coefficient: 0.357
```

```
n_clusters_ = len(set(db_labels)) - (1 if -1 in db_labels else 0)
n_noise_ = list(db_labels).count(-1)

Scene = dict(xaxis = dict(title  = 'PC1'),yaxis = dict(title  = 'PC2'),zaxis = dict(title  = 'PC3'))

trace = go.Scatter3d(x=tsne_3[:,0], y=tsne_3[:,1], z=tsne_3[:,2], mode='markers',marker=dict(color = db_labels, colorscale='rainbow', size =
layout = go.Layout(scene = Scene, height = 1000,width = 1000)
data = [trace]
fig = go.Figure(data = data, layout = layout)
```

```
fig.update_layout(title='DBSCAN clusters Derived from t-SNE', font=dict(size=12,))
fig.show()
```

Сохраним в общий датаферйм выделенные кластеры. Затем разделим на тренировочную (с метками) и тестовую часть (без меток), к тренировочной части по индексам обратно присоединим целевую переменную термостабильности 'tm'

```
#df_all['db_clusters'] = db_labels
df_all['ag4_clusters'] = ag4_labels
df_all.sample()
```

	('A', 'A')	('A', 'C')	('A', 'D')	('A', 'E')	('A', 'F')	('A', 'G')	('A', 'H')	('A', 'I')	('A', 'K')	('A', 'L')	...	K_count	M_count	F_co
id														
17076.0	-0.269121	0.445898	0.378115	-0.529316	0.636132	-0.069642	-0.320008	1.211632	0.216568	-0.341828	...	-0.136033	-0.17075	-0.631

1 rows × 445 columns

```
# посмотрим на распределение целевой переменной в кластерах
sns.displot(data=train, x='tm', hue='ag4_clusters', kde=True, alpha=.5, height=6, aspect=2, palette='rainbow') # , ax=ax[0]
# sns.displot(data=df, x='tm', hue=db_labels, kde=True, height=6, aspect=2, ax=ax[1])


# reset index and devide to two dataframes
# add tm clolumn
df_all.reset_index(inplace=True)


df_all.set_index('id', inplace=True)
train = df_all[df_all.index.isin(train_indexes)]
train.tail()


test = df_all[df_all.index.isin(test_indexes)]
test.head()


df_tm.set_index('seq_id', inplace=True)


train['tm'] = df_tm.tm
#df_train['protein_sequence'] = df_tm.protein_sequence
train.tail()
```

Поскольку вычисления на матрице эмбеддингов крайне ресурсозаратны, удалим колонки с эмбеддингами из финальной матрицы

```
# сохраняем результат
train.to_csv('train.csv')
test.to_csv('test.csv')


emb_cols = embeddings.columns.tolist()
emb_cols.remove('id')
train.drop(columns=emb_cols, inplace=True)
test.drop(columns=emb_cols, inplace=True)
```

https://radimrehurek.com/gensim/models/doc2vec.html

2.4 - Train/Test split, ресамплинг по pH

```
train = pd.read_csv('/content/train.csv')
```

Вспомним, что тренировочная выборка была сильно разбалансирована в части параметра кислтностного/щелочного pH. Учтем этот момент с помощью метода undersampling: снизим количество наблюдений с кислотным pH.

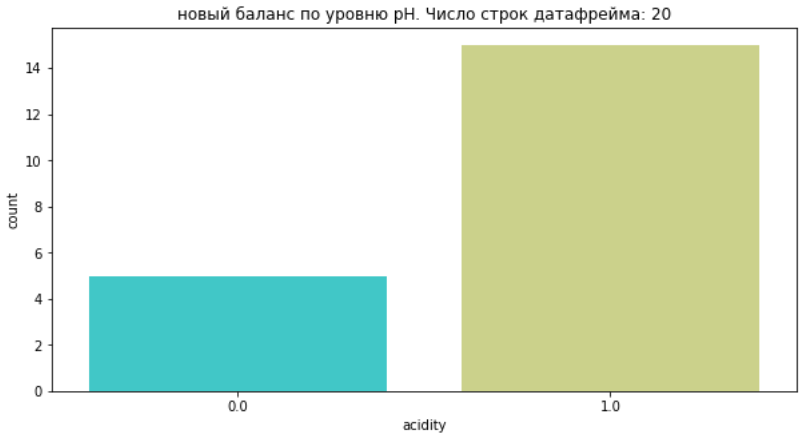
Сохраним итог в новый датафрейм

```
# Сохраним значение pH, разделяющее кислотные и щелочные наблюдения - в первом ряду

df_train[df_train.pH==7].pH[0]


train.pH[0]  # это масштабированное значение 7 pH, разделяющее кислотный и щелочные протеины -

     0.556048834628191


pH7 = train.pH[0]


# Щелочная часть датафрейма
df_base = train[train.pH>pH7]


# кислотная часть датафрейма
df_acid = train[train.pH<=pH7]
```

Случайным образом выделяем из датафрейма с кислотными pH часть в два раза больше, чем датафрейм с кислотными pH, чтобы баланс pH был 3 к 1. Это уже значительно лучше, чем 1 к 100 и позволит усилить фактор pH при этом меньше терять информации, если сравнять число наблюдений

```
n = (df_base.shape[0]*3) # число наблюдений
sample = df_acid.sample(n=n, random_state = 42)

#объединяем датасеты

df_resampled = pd.concat([df_base, sample])
size = df_resampled.shape[0]
# график распределения pH


plt.figure(figsize=(10,5))
sns.countplot('acidity', data = df_resampled, palette='rainbow')
plt.title('новый баланс по уровню pH. Число строк датафрейма: {}'.format(size))
```

```
Text(0.5, 1.0, 'новый баланс по уровню pH. Число строк датафрейма: 20')
```



Далее будем проверять и тестировать модели сразу на двух датасетах (до и после ресамплинга)

2.5 - Тестирование моделей

```
del df_resampled['Unnamed: 0']


mm=MinMaxScaler()


X = df_resampled.copy()
X.set_index('id', inplace=True)
y = X['tm']
del X['tm']
X = mm.fit_transform(X)


xr_train, xr_test, yr_train, yr_test = train_test_split(X, y, random_state=42, train_size=0.7, shuffle=True)
# для XGBoost
data_dmatrix = xgb.DMatrix(data=xr_train,label=yr_train)
print('Сбалансированный по pH:', xr_train.shape, xr_test.shape, yr_train.shape, yr_test.shape)
```

```
Сбалансированный по pH: (3760, 46) (1612, 46) (3760,) (1612,)
```

```
# Сравнение эффективностей моделей с параметрами по умолчанию

def run_models(models, x, y):
    stat = pd.DataFrame()
    cv = KFold(10)

# метрики из библиотеки: https://scikit-learn.org/stable/modules/model_evaluation.html#multimetric-scoring

    scoring = ['r2', # коэффициент детерминаци
```

```
                  'explained_variance',  # check
                   'max_error', # макс. ошибка остатки
                   'neg_mean_squared_error', # ~ MSE
                   'neg_mean_absolute_error'] # MAE
    for model_name, model in models.items():
        scores = cross_validate(model, x, y, cv=cv,  scoring=scoring)  #scores -> dict
        stat.loc[model_name, 'R2'] = scores['test_r2'].mean()
        stat.loc[model_name, 'Explained variance'] = scores['test_explained_variance'].mean()
        stat.loc[model_name, 'MSE'] = scores['test_neg_mean_squared_error'].mean()   # ~ MSE
        stat.loc[model_name, 'MAE'] = scores['test_neg_mean_absolute_error'].mean()  # ~MAE
        stat.loc[model_name, 'max_error'] = scores['test_max_error'].mean()
    return stat


models = {
    'LinearRegression': LinearRegression(),
    'Ridge': Ridge(),
    'Lasso': Lasso(),
    'SVR': SVR(),
    'KNeighborsRegressor': KNeighborsRegressor(),
    'DecisionTreeRegressor': DecisionTreeRegressor(random_state=42),
    'RandomForestRegressor': RandomForestRegressor(random_state=42),
    'XGBoost':  xgb.XGBRegressor()
}

result = run_models(models, xr_train, yr_train)
result.to_csv('result_resampled_default.csv')
result.style.highlight_max(axis=0)
```

```
[17:09:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:44] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:44] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[17:09:49] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

	R2	Explained variance	MSE	MAE	max_error
LinearRegression	0.241506	0.244482	-115.027385	-8.221764	-43.786964
Ridge	0.252826	0.255702	-113.401239	-8.179812	-39.331958
Lasso	-0.005589	0.000040	-152.518322	-9.351061	-43.495011
SVR	0.233128	0.252746	-117.057147	-7.554754	-40.921863
KNeighborsRegressor	0.495352	0.499262	-76.651730	-6.131244	-37.212000
DecisionTreeRegressor	0.217057	0.219561	-118.040821	-7.445441	-47.700000
RandomForestRegressor	0.600823	0.604039	-60.834265	-5.586593	-32.042100
XGBoost	0.536973	0.539802	-70.783064	-6.181169	-32.686181

```
# Оценим также перспективы использования полиноминальной регрессии

# Комментарий: GridSearch возвращает первую степень как оптимальную, то есть степенные модели не лучше обычной регрессии!

from sklearn.pipeline import make_pipeline

poly = PolynomialFeatures()
linear = LinearRegression()

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree), LinearRegression(**kwargs))

param_grid = {'polynomialfeatures__degree': np.arange(5), 'linearregression__fit_intercept': [True, False]}
poly_grid = GridSearchCV(PolynomialRegression(), param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search = poly_grid.fit(xr_train, yr_train)
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_
print(grid_search.best_estimator_)
print(best_accuracy)
print(best_parameters)
```

Лучшими моделями с параметрами по умолчанию по предварительной оценке являются:

- 1. RandomForestRegressor
- 2. XGBoost
- 3. KNeighborsRegressor

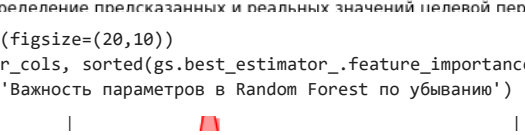
Далее попробуем подобрать для них наиболее оптимальные гиперпараметры и оценить, насколько они похволят повысить качество предсказаний

```
comparison_full = pd.DataFrame(columns={'Best_params', 'R2', 'MSE', 'MAE', 'MaxError'}, index={'RandomForestRegressor', 'XGBoost', 'KNeighbor
comparison_rescaled = pd.DataFrame(columns={'Best params', 'R2', 'MSE', 'MAE', 'MaxError'}, index={'RandomForestRegressor', 'XGBoost', 'KNeig
```

RandomForestRegressor

```
rf = RandomForestRegressor(random_state = 42)

param_grid = {
    'n_estimators': np.arange(100, 300, step=50), #количество деревьев в "лесу". Default: 100
    #'criterion': ['absolute_error', 'poisson'],  #метрика ошибки
    'max_depth': list(np.arange(5, 50, step=10)),  #глубина дерева
    'max_features': ['auto', 'sqrt', 'log2']  #randint(2, 4)
 }

# Random search of parameters, using 3 fold cross validation,
gs = GridSearchCV(estimator=rf, param_grid=param_grid, scoring='neg_mean_absolute_error',
                              cv = 10, verbose=0, n_jobs=-1,
                              return_train_score=True)

gs.fit(xr_train,yr_train)

best_estimator = gs.best_estimator_
best_params = gs.best_params_

yr_predict= gs.predict(xr_test)

print('Mean Absolute Error:', metrics.mean_absolute_error(yr_test, yr_predict))
print('Mean Squared Error:', metrics.mean_squared_error(yr_test, yr_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(yr_test, yr_predict)))
print('R2: ', metrics.r2_score(yr_test, yr_predict))

r2 = metrics.r2_score(yr_test, yr_predict)
mae = metrics.mean_absolute_error(yr_test, yr_predict)
mse = metrics.mean_squared_error(yr_test, yr_predict)
maxerror = metrics.max_error(yr_test, yr_predict)

# сохраняем в датафрейм лучшие гиперпараметры и метрики модели, полученные на их основе
comparison_rescaled.loc['RandomForestRegressor', 'Best params'] = str(best_params)
comparison_rescaled.loc['RandomForestRegressor', 'R2'] = r2
comparison_rescaled.loc['RandomForestRegressor', 'MSE'] = mse
comparison_rescaled.loc['RandomForestRegressor', 'MAE'] = mae
comparison_rescaled.loc['RandomForestRegressor', 'MaxError'] = maxerror
comparison_rescaled.to_csv('comparison_rescaled.csv')

# построим график, чтобы оценить результат визуально
sns.distplot(yr_test, hist=True, kde=True, color = 'blue', hist_kws={'edgecolor':'black'})
sns.distplot(yr_predict, hist=True, kde=True, color = 'red')
plt.title('Распределение предсказанных и реальных значений целевой переменной tm')
```

```
Text(0.5, 1.0, 'Распределение предсказанных и реальных значений целевой переменной tm')
```

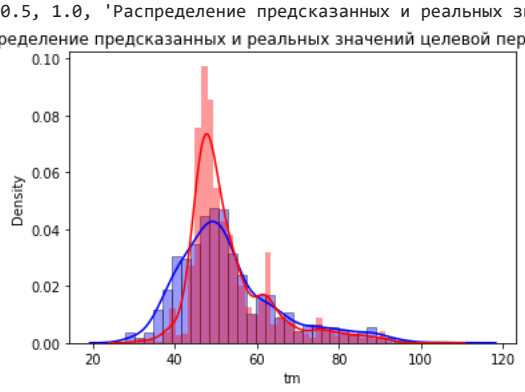



```
plt.figure(figsize=(20,10))
plt.barh(xr_cols, sorted(gs.best_estimator_.feature_importances_))
plt.title('Важность параметров в Random Forest по убыванию')
```

XGBoost

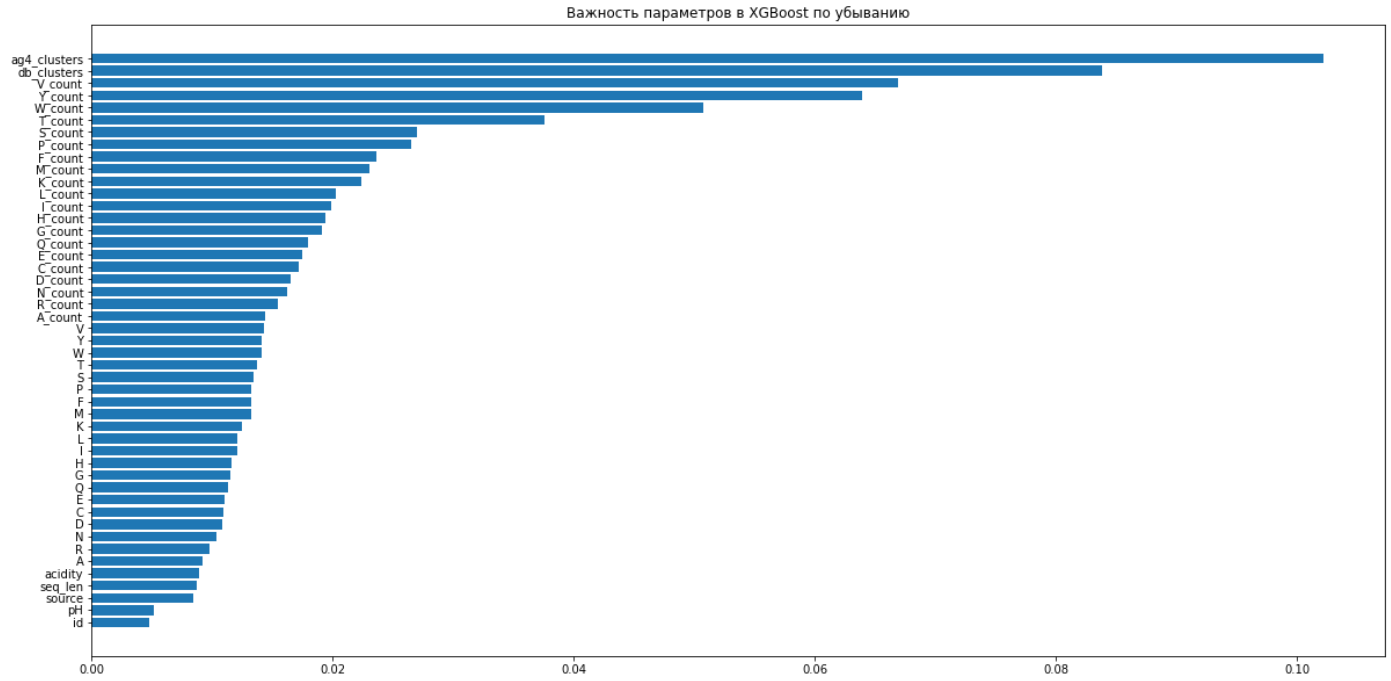

```
xgb_model = XGBRegressor()

params = {
      "colsample_bytree": list(np.arange(0.3, 0.8, 0.1)), # часть колонок
      "gamma": list(np.arange(0, 0.3, 0.1)), # минимальная потеря для следующего сплита
      "learning_rate": list(np.arange(0.05, 0.31, 0.05)), # default 0.1
      "n_estimators": [100, 150], # default 100
      "subsample": list(np.linspace(0.01, 1.0, 10)) # % строк в тренировочном сампле
}

gs = GridSearchCV(xgb_model, params, cv=5, n_jobs=5, return_train_score=True)
gs.fit(xr_train, yr_train)

print(gs.best_score_)
print(gs.best_params_)

print('Mean Absolute Error:', metrics.mean_absolute_error(yr_test, yr_predict))
print('Mean Squared Error:', metrics.mean_squared_error(yr_test, yr_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(yr_test, yr_predict)))
print('R2: ', metrics.r2_score(yr_test, yr_predict))

best_params = gs.best_params_
r2 = metrics.r2_score(yr_test, yr_predict)
mae = metrics.mean_absolute_error(yr_test, yr_predict)
mse = metrics.mean_squared_error(yr_test, yr_predict)
maxerror = metrics.max_error(yr_test, yr_predict)

yr_predict = gs.predict(xr_test)

# сохраняем в датафрейм лучшие гиперпараметры и метрики модели, полученные на их основе
comparison_rescaled.loc['XGBoost', 'Best params'] = str(best_params)
comparison_rescaled.loc['XGBoost', 'R2'] = r2
comparison_rescaled.loc['XGBoost', 'MSE'] = mse
comparison_rescaled.loc['XGBoost', 'MAE'] = mae
comparison_rescaled.loc['XGBoost', 'MaxError'] = maxerror

# построим график, чтобы оценить результат визуально
sns.distplot(yr_test, hist=True, kde=True, color = 'blue', hist_kws={'edgecolor':'black'})
sns.distplot(yr_predict, hist=True, kde=True, color = 'red')
plt.title('Распределение предсказанных и реальных значений целевой переменной tm')
```

```
Text(0.5, 1.0, 'Распределение предсказанных и реальных значений целевой переменной tm')
```




```
t=train.copy()
del t['tm']
plt.figure(figsize=(20,10))
plt.barh(t.columns, sorted(gs.best_estimator_.feature_importances_))
plt.title('Важность параметров в XGBoost по убыванию')
```

```
Text(0.5, 1.0, 'Важность параметров в XGBoost по убыванию')
```



Регрессор K-ближайщих соседей

```
KNmodel = KNeighborsRegressor()

cv = KFold(n_splits=3, shuffle=True)

params = {
    'n_neighbors': range(2, 30, 2), 'weights': ['uniform','distance']}

gs = GridSearchCV(KNmodel, params, cv=5, n_jobs=5, return_train_score=True)
gs.fit(xr_train, yr_train)

print(gs.best_score_)
print(gs.best_params_)

print('Mean Absolute Error:', metrics.mean_absolute_error(yr_test, yr_predict))
print('Mean Squared Error:', metrics.mean_squared_error(yr_test, yr_predict))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(yr_test, yr_predict)))
print('R2: ', metrics.r2_score(yr_test, yr_predict))

best_params = gs.best_params_
r2 = metrics.r2_score(yr_test, yr_predict)
mae = metrics.mean_absolute_error(yr_test, yr_predict)
mse = metrics.mean_squared_error(yr_test, yr_predict)
maxerror = metrics.max_error(yr_test, yr_predict)

yr_predict = gs.predict(xr_test)

# сохраняем в датафрейм лучшие гиперпараметры и метрики модели, полученные на их основе
comparison_rescaled.loc['KNeighborsRegressor', 'Best params'] = str(best_params)
comparison_rescaled.loc['KNeighborsRegressor', 'R2'] = r2
comparison_rescaled.loc['KNeighborsRegressor', 'MSE'] = mse
comparison_rescaled.loc['KNeighborsRegressor', 'MAE'] = mae
comparison_rescaled.loc['KNeighborsRegressor', 'MaxError'] = maxerror

# построим график, чтобы оценить результат визуально
sns.distplot(yr_test, hist=True, kde=True, color = 'blue', hist_kws={'edgecolor':'black'})
sns.distplot(yr_predict, hist=True, kde=True, color = 'red')
plt.title('Распределение предсказанных и реальных значений целевой переменной tm')
```

```
0.5596078725256776
{'n_neighbors': 8, 'weights': 'distance'}
Mean Absolute Error: 5.442780690157679
Mean Squared Error: 55.92339121662936
Root Mean Squared Error: 7.478194382110521
R2:  0.6548340976907816
Text(0.5, 1.0, 'Распределение предсказанных и реальных значений целевой переменной tm')
```

3 - Нейронная сеть

```
xr_train.shape, yr_train.shape, xr_test.shape, yr_test.shape
```

```
((3760, 47), (3760,), (1612, 47), (1612,))
```

```
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
tf.autograph.set_verbosity(0)
```

```
tf.random.set_seed(42)  # applied to achieve consistent results
NNmodel = Sequential(
    [
        tf.keras.Input(shape=(46,)),
        Dense(46, activation='relu', kernel_regularizer=regularizers.L1(0.01), name = 'layer1'), # added lambda as predictions were flat
        Dense(138, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer2'),
        Dense(138, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer3'),
        Dense(92, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer4'),
        Dense(92, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer5'),
        Dense(46, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer6'),
        Dense(46, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer7'),
        Dense(1, activation='linear', kernel_regularizer=regularizers.L1(0.01), name = 'layer8')
    ])

NNmodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 46)                2162

 layer2 (Dense)              (None, 138)               6486

 layer3 (Dense)              (None, 138)               19182

 layer4 (Dense)              (None, 92)                12788

 layer5 (Dense)              (None, 92)                8556

 layer6 (Dense)              (None, 46)                4278

 layer7 (Dense)              (None, 46)                2162

 layer8 (Dense)              (None, 1)                 47

=================================================================
Total params: 55,661
Trainable params: 55,661
Non-trainable params: 0
_________________________________________________________________
```

```
tf.random.set_seed(42)  # applied to achieve consistent results
NNmodel = Sequential(
    [
        tf.keras.Input(shape=(46,)),
```

```
        Dense(46, activation='relu', kernel_regularizer=regularizers.L1(0.01), name = 'layer1'), # added lambda as predictions were flat
        Dense(138, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer2'),
        Dense(138, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer3'),
        Dense(92, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer4'),
        Dense(92, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer5'),
        Dense(46, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer6'),
        Dense(46, activation='relu',  kernel_regularizer=regularizers.L1(0.01), name = 'layer7'),
        Dense(1, activation='linear', kernel_regularizer=regularizers.L1(0.01), name = 'layer8')
    ])

NNmodel.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 46)                2162

 layer2 (Dense)              (None, 138)               6486

 layer3 (Dense)              (None, 138)               19182

 layer4 (Dense)              (None, 92)                12788

 layer5 (Dense)              (None, 92)                8556

 layer6 (Dense)              (None, 46)                4278

 layer7 (Dense)              (None, 46)                2162

 layer8 (Dense)              (None, 1)                 47

=================================================================
Total params: 55,661
Trainable params: 55,661
Non-trainable params: 0
_________________________________________________________________
```

```
NNmodel.compile(loss='mean_absolute_error',       # MAE is less sensitive to outliers
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001))

history = NNmodel.fit(xr_train, yr_train,
    epochs=500, verbose=0, validation_split = 0.2)  #verbose - supress logging

yr_predict = NNmodel.predict(xr_test)

r2 = metrics.r2_score(yr_test, yr_predict)
mae = metrics.mean_absolute_error(yr_test, yr_predict)
mse = metrics.mean_squared_error(yr_test, yr_predict)
maxerror = metrics.max_error(yr_test, yr_predict)

# сохраняем в датафрейм метрики модели

comparison_rescaled.loc['NeuralNetwork_1', 'R2'] = r2
comparison_rescaled.loc['NeuralNetwork_1', 'MSE'] = mse
comparison_rescaled.loc['NeuralNetwork_1', 'MAE'] = mae
comparison_rescaled.loc['NeuralNetwork_1', 'MaxError'] = maxerror

# оценим график функции потерь (в т.ч. на валидационной части выборки)

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
def plot_loss(history):
    plt.figure(figsize=(5,5))
    plt.plot(history.history['loss'], label='потеря')
    plt.plot(history.history['val_loss'], label='валидационная потеря', c='r')
    #plt.ylim([0, 2])
    plt.xlabel('Эпохи')
    plt.ylabel('Потеря')
    plt.legend()
    plt.grid(True)
```


```
NNmodel.save('NNmodel.mlp')
```

```
# построим график, чтобы оценить результат визуально
sns.distplot(yr_test, hist=True, kde=True, color = 'blue', hist_kws={'edgecolor':'black'})
sns.distplot(yr_predict, hist=True, kde=True, color = 'red')
plt.title('Предсказанные значения сгруппировались вокруг средних начений')
plt.show()
```

Финальная свободная таблица с метриками моделей

```
# Посмотрим на итоговые результаты сравнения моделей после подбора гиперпараметров
comparison_rescaled
```

	Unnamed: 0	Best params	R2	MaxError	MAE	MSE
0	KNeighborsRegressor	{'n_neighbors': 8, 'weights': 'distance'}	0.65	34.96	5.44	55.92
1	XGBoost	{'colsample_bytree': 0.7000000000000002, 'gamm...	0.64	39.44	5.43	57.65
2	RandomForestRegressor	{'max_depth': 45, 'max_features': 'auto', 'n_...	0.64	39.44	5.43	57.65
3	NeuralNetwork	NaN	0.46	54.40	6.65	87.08

Наилучшие результаты по предсказанию термостабильности показал алгоритм KNN с достаточно высоким уровнем R2 и самыми низкими ошибками MAE/MSE/Max Error