

```
from os import listdir
from time import localtime
from os.path import join

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns

%matplotlib inline
```

```
mpl.rcParams.update({'font.size': 22})
```

## TOC

1. Краткое подведение итогов
2. Анализ данных
  - i. Методы каротажа
    - a. Методы электрического каротажа
      - a. Группа методов кажущегося сопротивления (KC) собственно
      - b. Группа электромагнитных методов
      - c. Группа методов электрохимической активности
    - b. Методы радиоактивного каротажа
      - a. Группа гамма-методов
  - ii. Utility fuctions
  - iii. Загрузка данных
    - d. Загрузка расшифровки литологии
    - e. Загрузка скважин
  - iv. Анализ скважин
    - a. Графики корреляций методов в различных скважинах
    - b. Куст скважин
    - c. Распределения переменных
    - d. Анализ гистограм интервалов относительно значений целевой переменной
    - e. Распределения целевой переменной
3. Обучение
  - i. Baseline algorithm (count classifier)

- ii. Разбиение на train/dev/test sets
  - a. Отсутвующие методы
  - b. Выборка без отсутвующих методов
  - c. Обучение методов
    - a. Random Forest
    - b. LogisticRegression
    - c. Count classifier
  - d. Выводы:
  - d. Заполнить пропущенные методы и значения средним по датасету
    - a. Обучение методов
      - a. Random Forest
      - b. LogisticRegression
  - e. Заполнить пропущенные методы и значения "нулями"
    - a. Обучение методов
      - b. Random Forest
      - c. LogisticRegression
  - f. Заполнение пропущенных значений аппроксимацией
    - a. Построение аппроксимирующей модели
    - b. Train/test split
    - c. Обучение методов
      - a. Random Forest
      - b. LogisticRegression
    - d. Выводы:

#### 4. Изучение важности KS

##### i. Классификация

Краткое подведение итогов:

- На данном этапе, лучше всего работает отбрасывание отсутствующих значений.
- Различные методы замещения "нuleй" работают плохо.
  - На что стоит обратить внимание:
    - **Выборка без отсутвующих методов**
      - Результаты классификации с помощью Random Forest и Logistic Regression показывают, что необходимо использовать понижение количества наблюдаемых занчений (Under sampling)
      - Стоит добавить:
        - Кросс валидацию с one deposit out;

- Under sampling
  - Попробовать альтернативные методы классификации
- 
- **Заполнить пропущенные методы и значения средним по датасету**
    - Заполнение средними значениями дает не очень хорошие результаты. Скорее всего, дело в том, что:
      - на разной глубине разные значения методов;
      - Мы добавляем еще больше лишних значений (в частности песка), поэтому разумно было бы использовать undersampling
  - **Заполнить пропущенные методы и значения "нулями"**
    - Заполнение "нулями" вообще не работает хд
    - Поэтому для них надо искать замену либо вообще их удалять

- **Заполнение пропущенных значений аппроксимацией**

К сожалению, магия не случилась и точность не повысилась.

Но тут может быть другая проблема быть причиной:

- При моделировании методов, коэффициент детерминации во многих случаях очень маленький, поэтому имеет смысл обратить на это внимание.
  - Поиграть с масштабом данных;
  - Попробовать иные методы нелинейной регрессии.
- Мы также добавляем еще больше лишних значений для и так большого количества лягушек, мб в этом проблема.

Итог такой: Данные в любом случае надо чистить. Поскольку только избавление от пропущенных значений повышает точность. Имеет смысл также попробовать визуализировать данные после заполнения с помощью TSNE. Также стоит, как уже было неоднократно сказано, применить undersampling.

## Анализ данных

---

### Методы каротажа

---

[wiki\\_link](#)

# Методы электрического каротажа

## Группа методов кажущегося сопротивления (КС) собственно

- Единица ПС измерения – милливольт (мВ)
- **КС** – кажущееся сопротивление с нефокусированными зондами. Самый распространённый метод данной группы, являющийся скважинным аналогом метода электрического профилирования в электроразведке
- **резистивиметрия (РЕЗ)**. С помощью этого метода измеряют удельное электрическое сопротивление жидкости, заполняющей в данный момент скважину. Жидкость может быть представлена как буровым раствором (его сопротивление заранее известно), так и пластовыми флюидами (нефть, пресная или минерализованная вода), а также их смесью
- **БКЗ** – боковое каротажное зондирование. Данный метод является скважинным аналогом метода вертикального электрического зондирования в электроразведке
- **микрокаротаж** – разновидность КС с зондами очень малого размера, в плотную прижимаемыми к стенкам скважины. С помощью данного метода, преимущественно, ищут только коллекторы по скважине
- **БК** – боковой каротаж. Отличие от классического КС заключается в фокусировке тока зондом
- **МБК** – микробоковой каротаж. Отличие данного метода от микрокаротажа заключается в фокусировке тока зондом

## Группа электромагнитных методов

Основное преимущество данной группы методов заключается в том, что их возможно использовать в сухих скважинах, не заполненных токопроводящим буровым раствором. Кроме того, его возможно применять и в скважинах, заполненных буровым раствором на основе нефти, которые тоже не проводят постоянный электрический ток.

Встречаются следующие разновидности:

- **ИК** – индукционный каротаж. При проведении используют сравнительно низкие частоты – до 200 кГц

индукционный каротаж на высоких частотах, результаты которого зависят как от электропроводности пород, так и от их диэлектрической проницаемости:

- **ВМП** – волновой метод проводимости с частотой 1-5 МГц
- **ВДК** – волновой диэлектрический каротаж с частотой до 60 МГц
- **ВЭМКЗ** - высокочастотное электромагнитное каротажное зондирование
- **ВИКИЗ** – высокочастотное индукционное каротажное изопараметрическое зондирование. Метод является аналогом БКЗ, но вместо постоянного тока

используется переменный

## Группа методов электрохимической активности

- **ПС** — метод самопроизвольной поляризации, также известный как метод потенциалов самопроизвольной поляризации. Является скважинным аналогом метода естественного поля в электроразведке
- **ЭК** — электролитический каротаж. Скважинный аналог метода вызванной поляризации в электроразведке
- **МЭП** — метод электродных потенциалов. Данный метод существует исключительно в скважинном варианте и не имеет аналогов в полевой электроразведке

## Методы радиоактивного каротажа

### Группа гамма-методов

- **ГК** — гамма-каротаж. Очень простой и распространённый метод, измеряющий только естественное гамма-излучение от пород, окружающих скважину. Существует его чуть более усложнённый вариант — спектрометрический гамма-каротаж (СГК или ГК-С), который позволяет различить попавшие в детектор геофизического зонда гамма-кванты по их энергии. По этому параметру можно точнее судить о характере слагающих толщу пород.
  - Основная расчетная величина – мощность экспозиционной дозы в микрорентгенах в час (МЭД, мкР/ч). Измеряемая величина определяется концентрацией, составом и пространственным распределением ЕРЭ, плотностью ρ<sub>р</sub> и эффективным атомным номером Z<sub>эфф</sub> пород. Гамма-каротаж
  - Входит в число обязательных методов
- 
- **ГГК** — гамма-гамма каротаж. Геофизический зонд облучает породу гамма-излучением, в результате которого порода становится радиоактивной и в ответ тоже излучает гамма-кванты. Именно эти кванты и регистрируются зондом. Существует две основных разновидности метода:
  - плотностная — ГГК-П (иногда встречается обозначение ПГГК)
  - селективная — ГГК-С (может обозначаться как Z-ГГК, С-ГГК и т. п.)
- **РРК** — рентгенорадиометрический каротаж. Его название формально не соответствует общепринятой системе, поэтому иногда встречается название ГРК (гамма-рентгеновский каротаж), но РРК является общеупотребимым.

## Utility fuctions

---

```

def load_dvfs(path='../../data/digdes-data/coal-wells-2/', zeros=-999.25):
    """
    Load wells as a dict with their file names
    """
    csvs = [file for file in listdir(path) if file.endswith('.csv')]
    dtfs = dict()
    for csv in csvs:
        dtf = pd.read_csv(path + csv, sep=';', decimal=',', encoding='cp1251')
        dtf[dtf == zeros] = np.nan
        dtf[dtf == -9999] = np.nan
        if 'REZ' in dtf.columns:
            dtf.REZ[dtf.REZ < 0] = np.nan
        dtf.LIT
        dtfs[csv[:-4]] = dtf
    return dtfs

```

```

## Try to convert data to float to find bullshit
def check_values_multiple_comas(dvfs, column):
    """
    Pass dictionary of dataframes to find bad dataframe and value
    """
    for name, dtf in dvfs.items():
        for val in dtf[column]:
            try:
                if type(val) == str:
                    float(val.replace(',', '.'))
            except:
                print(name, val)

```

```

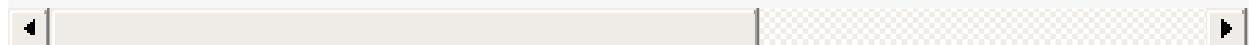
def check_values_for_str(dtf, column, name):
    for val in dtf[column]:
        try:
            if type(val) == str:
                float(val)
        except:
            print(name, val)

```

```

def generate_file_name(extension, f_name=''):
    """generate file name with `extension`"""
    tm = localtime()
    name = f_name + '-' + '-'.join(map(str, [tm.tm_year, tm.tm_mon, tm.tm_mday, tm.
    return name

```



```

def plot_conusion_matrix(matrix, ticks='auto', normalized=True, plot=True,
                        save_fig=True, save_path='', file_name='', extension='.png',
                        **kwargs):
    """
    function to plot, print and save confusion matrix
    :params:
        matrix      -- calculated confusion matrix
        ticks       -- class labels
        normalized -- (bool) normalize or not matrix
        plot        -- (bool) plot or print matrix
        save_fig   -- (bool) save figure or not to `save_path`
        save_path  -- (str) path to save figure. if None, saves to directory of fun
        file_name  -- (str) name of figure with extension
        **kwargs    -- parameters for sns.heatmap
    """
    if normalized:
        mat = matrix/(np.sum(matrix, axis=1).reshape(-1, 1))
    else:
        mat = matrix
    if plot:
        sns.heatmap(mat, annot=True, xticklabels=ticks, yticklabels=ticks, **kwargs)
        plt.tight_layout()
        if save_fig:
            name = generate_file_name(extension, file_name)
            plt.savefig(join(save_path, name))
    else:
        print(mat)

```



```

def plot_correlations_in_well(dtf, name, axis=None):
    sns.heatmap(dtf.corr(), annot=True, ax=axis)
    if axis:
        axis.set_title('dtf_{}'.format(name))
    else:
        plt.title('dtf_{}'.format(name))

```

```

def indices_with_swaps(shape):
    for i in range(shape[0]):
        for j in range(shape[1]):
            yield i, j

```

```

def plot_correlations_in_deposit(dtf_dict, figsize=(20, 20)):
    plot_shape = (int(np.ceil(np.sqrt(len(dtf_dict)-1))),
                  int(np.ceil(np.sqrt(len(dtf_dict)-1))))
    indices = indices_with_swaps(plot_shape)
    fig, axis = plt.subplots(plot_shape[0], plot_shape[1], figsize=figsize, sharex=

```

```
for name, dtf in dtf_dict.items():
    ind = next(indices)
    plot_correlations_in_well(dtf, name, axis[ind[0]], ind[1]))
```



## Загрузка данных

### Загрузка расшифровки литологий

```
data_source = 'data/digdes-data/'
```

```
lithology = pd.read_excel(data_source + 'coal-wells-1/Литология.xlsx')
lithology.index = lithology['Литология'].values
lithology.drop(['Литология'], inplace=True, axis=1)
lithology = lithology.to_dict()['порода ']
```

```
lithology
```

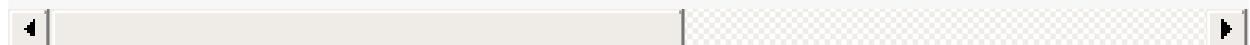
```
{1: 'песчаник',
 2: 'Аргиллит алевритовый',
 3: 'Аргиллит',
 4: 'Гравелит',
 5: 'алевролит',
 6: 'Аргиллит углистый',
 7: 'Уголь',
 8: 'Супесь',
 9: 'Суглинок',
 10: 'Валунно-галечные отложения',
 11: 'песок'}
```

```
def print_value_counts_maped(dtf, val):
    for v, k in dtf[val].value_counts().items():
        if v in lithology.keys():
            print('{} : {}'.format(lithology[int(v)], k))
        else:
            print('{} : {}'.format(v, k))
```

## Загрузка скважин

```
## Loading dataframes
dtfs_first = load_dtfs(data_source + 'coal-wells-2/first-bath/')
dtfs_second = load_dtfs(data_source + 'coal-wells-2/second-bath/')
dtfs_third = load_dtfs(data_source + 'coal-wells-2/third-bath/')

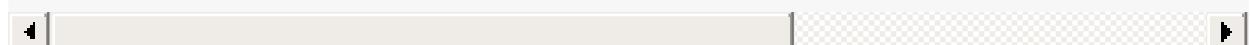
print('Количество скважин в первом, во втором и третьем месторождении: {}, {}, {}'.format(dtfs_first.shape[0], dtfs_second.shape[0], dtfs_third.shape[0]))
```



```
C:\Program Files\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame
```

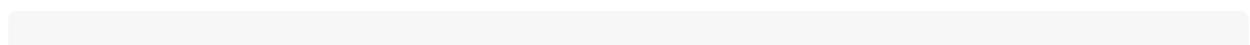
```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#inplace-evaluation
if sys.path[0] == '':
    del sys.path[0]
```

```
Количество скважин в первом, во втором и третьем месторождении: 24, 28, 164
```



```
dtf_first.LIT.
```

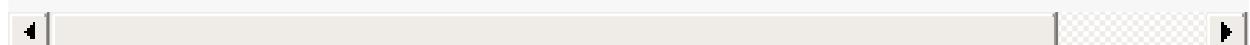
```
## Concatenating dataframes
dtf_first = pd.concat([d for d in dtfs_first.values()], ignore_index=True)
dtf_second = pd.concat([d for d in dtfs_second.values()], ignore_index=True)
dtf_third = pd.concat([d for d in dtfs_third.values()], ignore_index=True)
```



```
print('Количество данных в первом месторождении с отсутствующими значениями {} и без {}')
print('Количество данных во втором месторождении с отсутствующими значениями {} и без {}')
print('Количество данных во третьем месторождении с отсутствующими значениями {} и без {}')
```



```
Количество данных в первом месторождении с отсутствующими значениями 352137 и без 352137
Количество данных во втором месторождении с отсутствующими значениями 170951 и без 170951
Количество данных во третьем месторождении с отсутствующими значениями 1225094 и без 1225094
```



В первом месторождении столько значений пропадает из-за того, что в некоторых скважинах отсутствуют некоторые методы каротажа

```
print('Методы каротажа в первом и во втором месторождении')
for f, s, t in zip(sorted(dtf_first.columns), sorted(dtf_second.columns), sorted(dt
    print(f, s, t)
```



```
Методы каротажа в первом и во втором месторождении
BK BK DS
DS DS Dept
Dept DT GGK(p)
GGK(p) Dept GGK(s)
GR GGK(p) GR
KS GR KS
LIT KS LIT
REZ LIT PS
```

Месторождения различаются в два метода:

- В первом присутствует KS
- Во втором DT

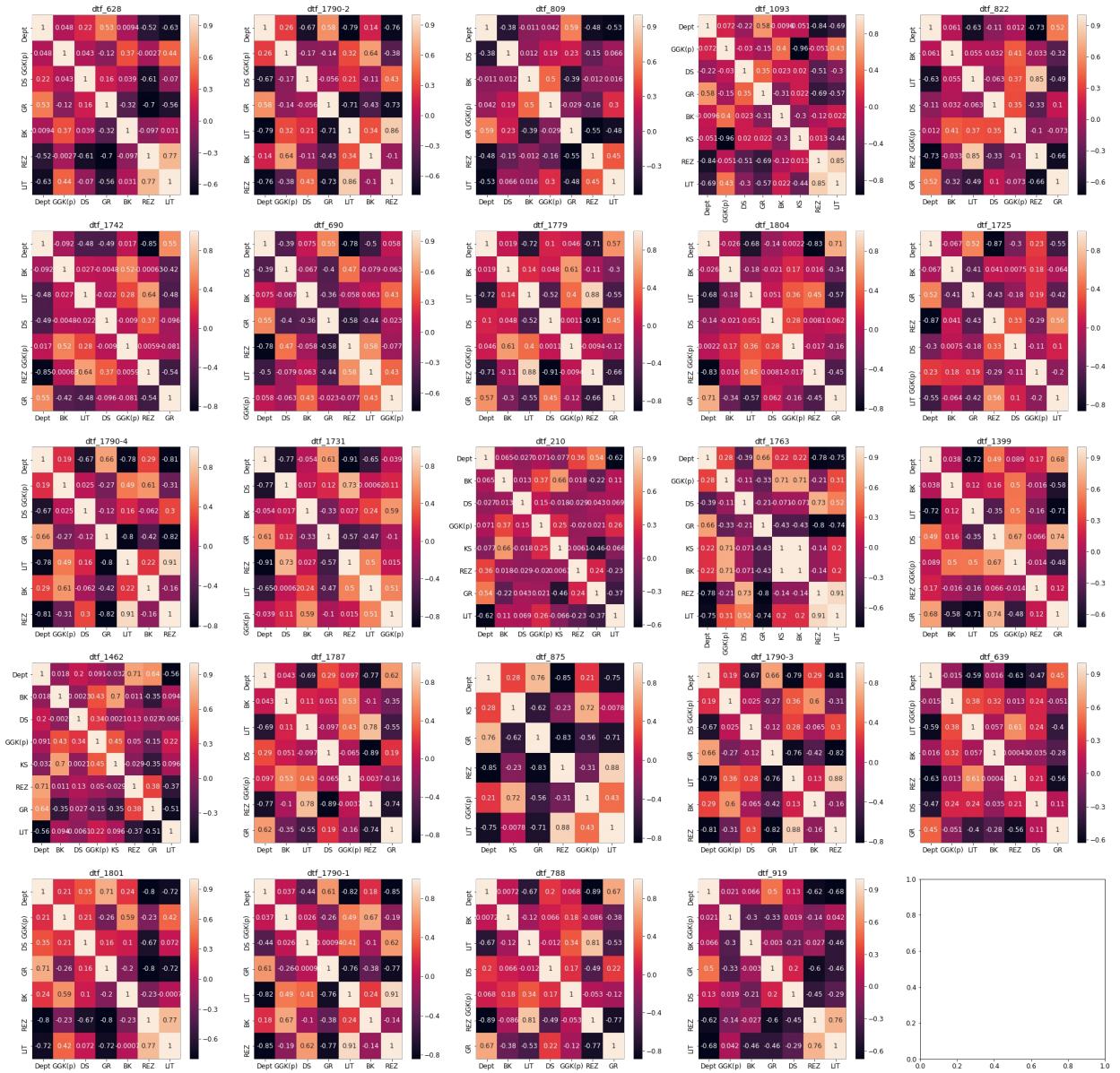
## Анализ скважин

### Графики корреляций методов в различных скважинах

Корреляции в первом меторождении

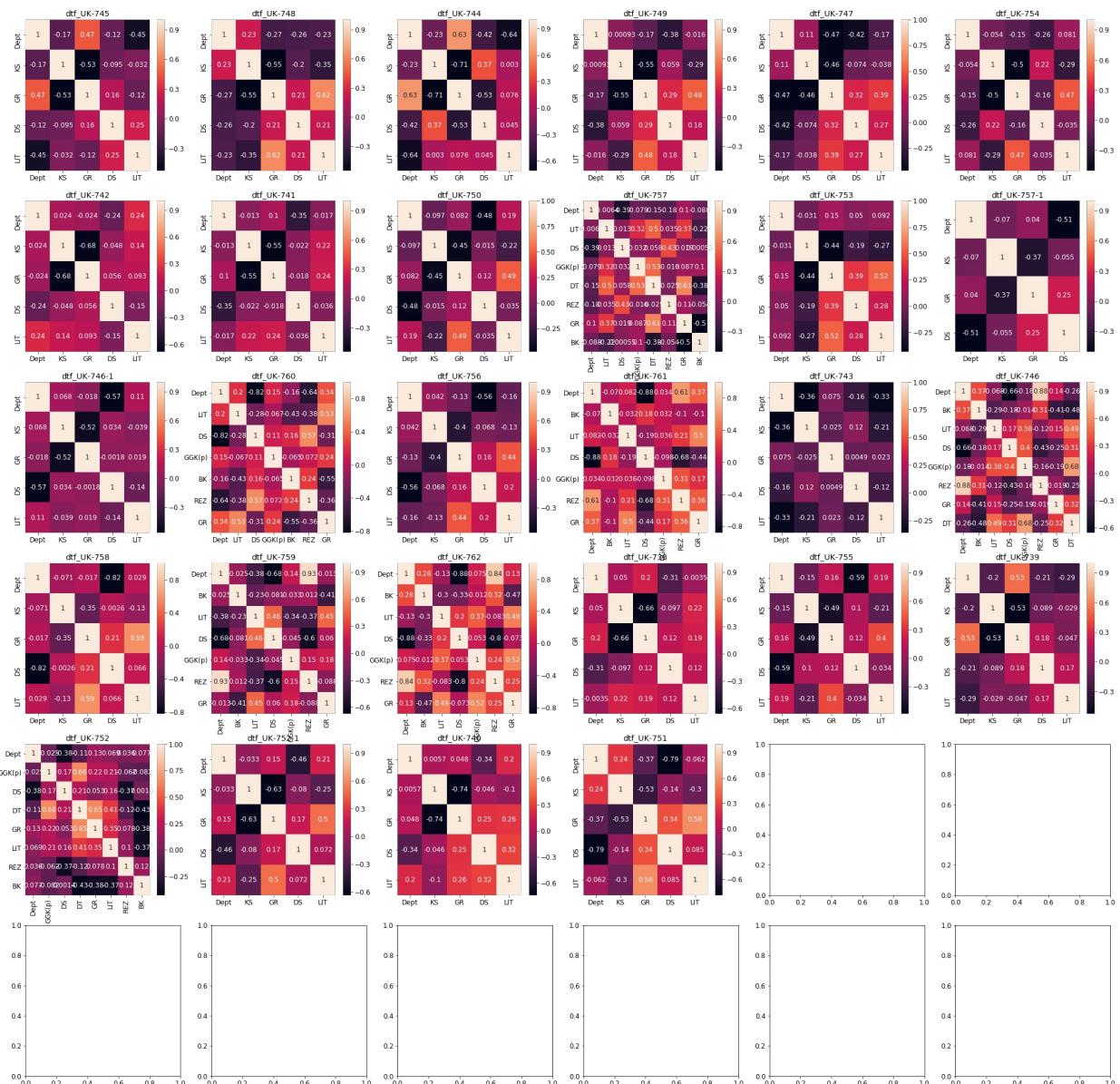
```
mpl.rcParams.update({'font.size': 12})
```

```
plot_correlations_in_deposit(dtsfs_first, figsize=(35, 35))
```



Корреляции во втором месторождении

```
plot_correlations_in_deposit(dtfs_second, figsize=(35, 35))
```

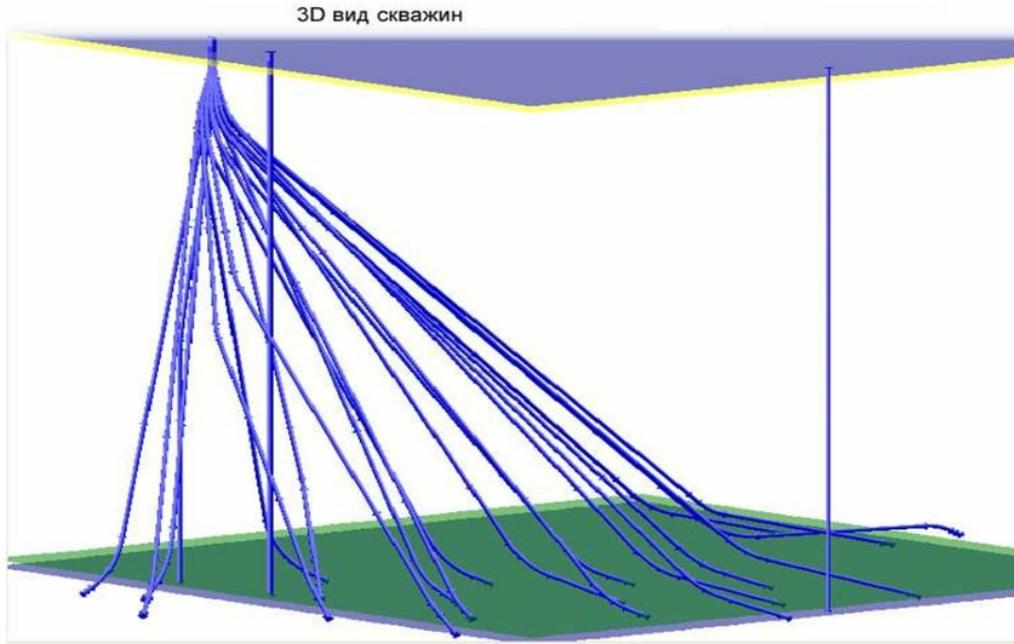


- В Первом месторождении в скважине 875 не показывает корреляцию LIT

# Куст скважин

wiki link

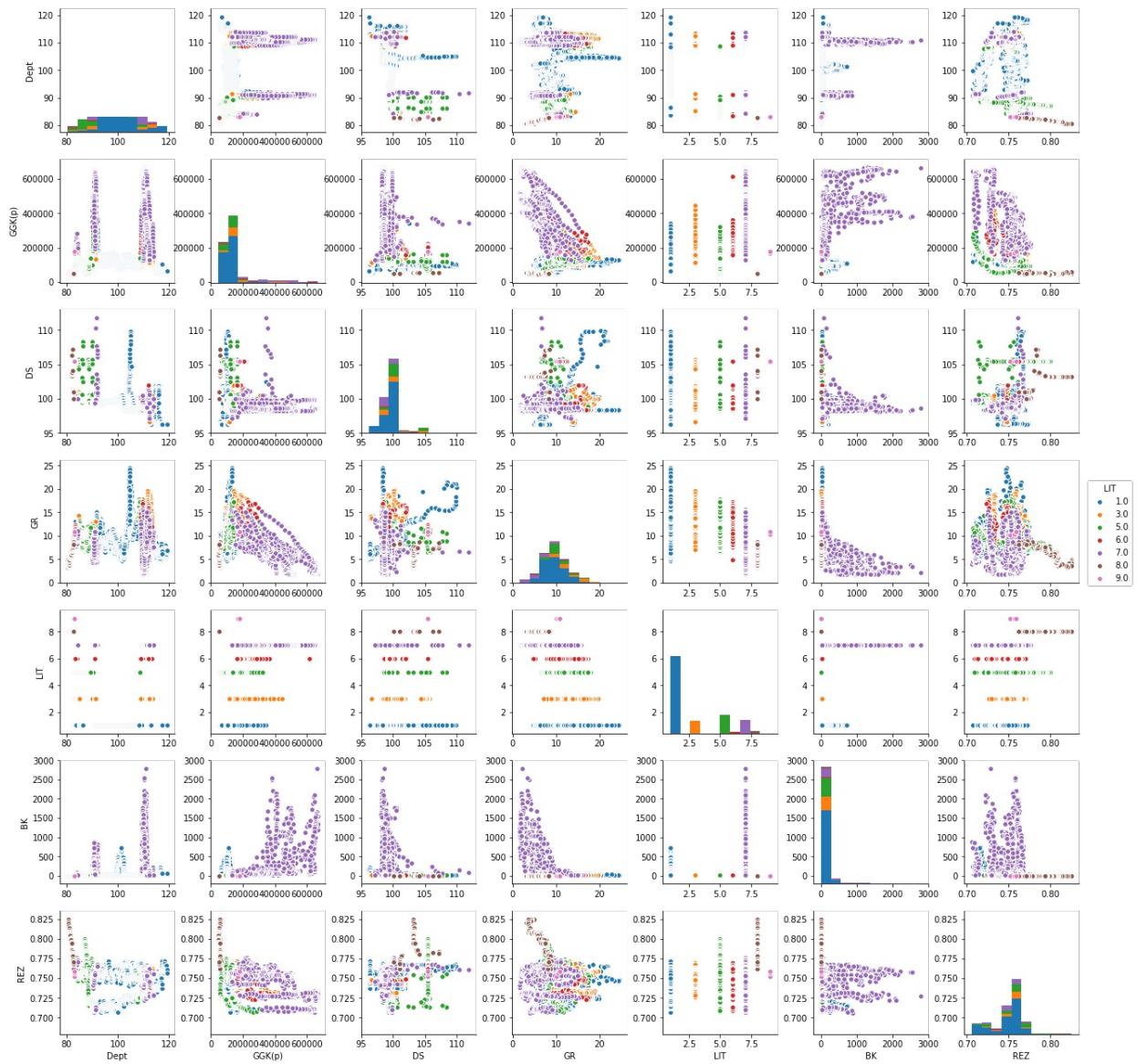
# Куст скважин



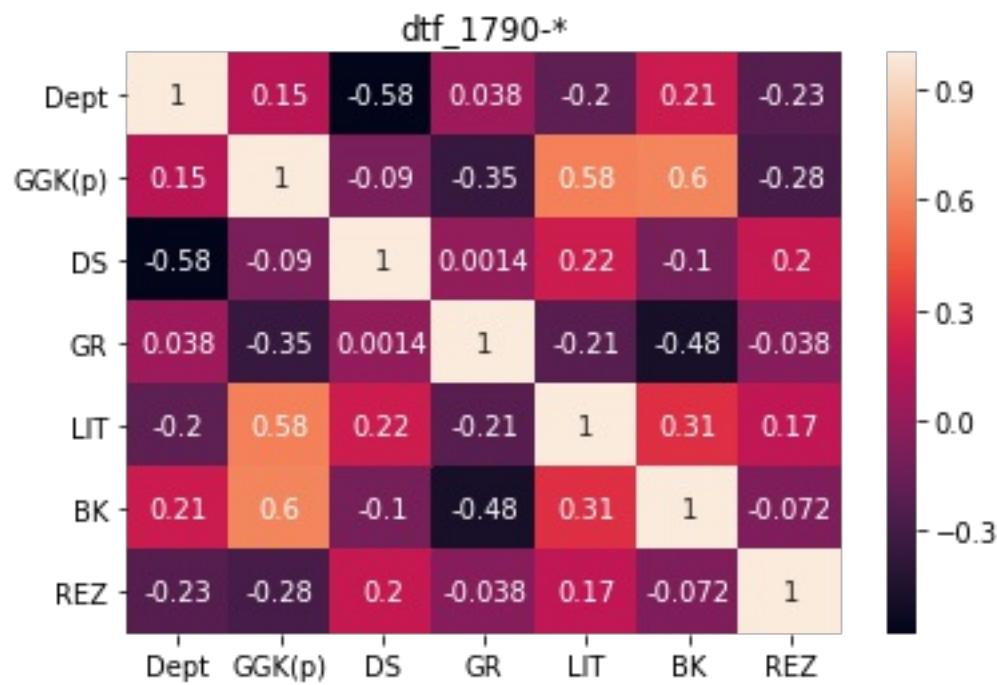
Скважины 1790-\* -- образуют куст скважин. Как видно на изображении, не сильно отдалены, поэтому, можно предположить, что данные в этих скважинах будут коррелировать

```
sns.pairplot(pd.concat([dtfs_first['1790-1'].dropna(), dtfs_first['1790-2'].dropna(),
                      dtfs_first['1790-3'].dropna(), dtfs_first['1790-4'].dropna(),
                      ignore_index=True], hue='LIT')
```

```
<seaborn.axisgrid.PairGrid at 0x1180e1da0>
```



```
plot_correlations_in_well(pd.concat([dtfs_first['1790-1'].dropna(), dtfs_first['1790-2'].dropna(), dtfs_first['1790-3'].dropna(), dtfs_first['1790-4'].dropna(), dtfs_first['1790-5'].dropna(), dtfs_first['1790-6'].dropna(), dtfs_first['1790-7'].dropna()], ignore_index=True),
                           name='1790-*')
```

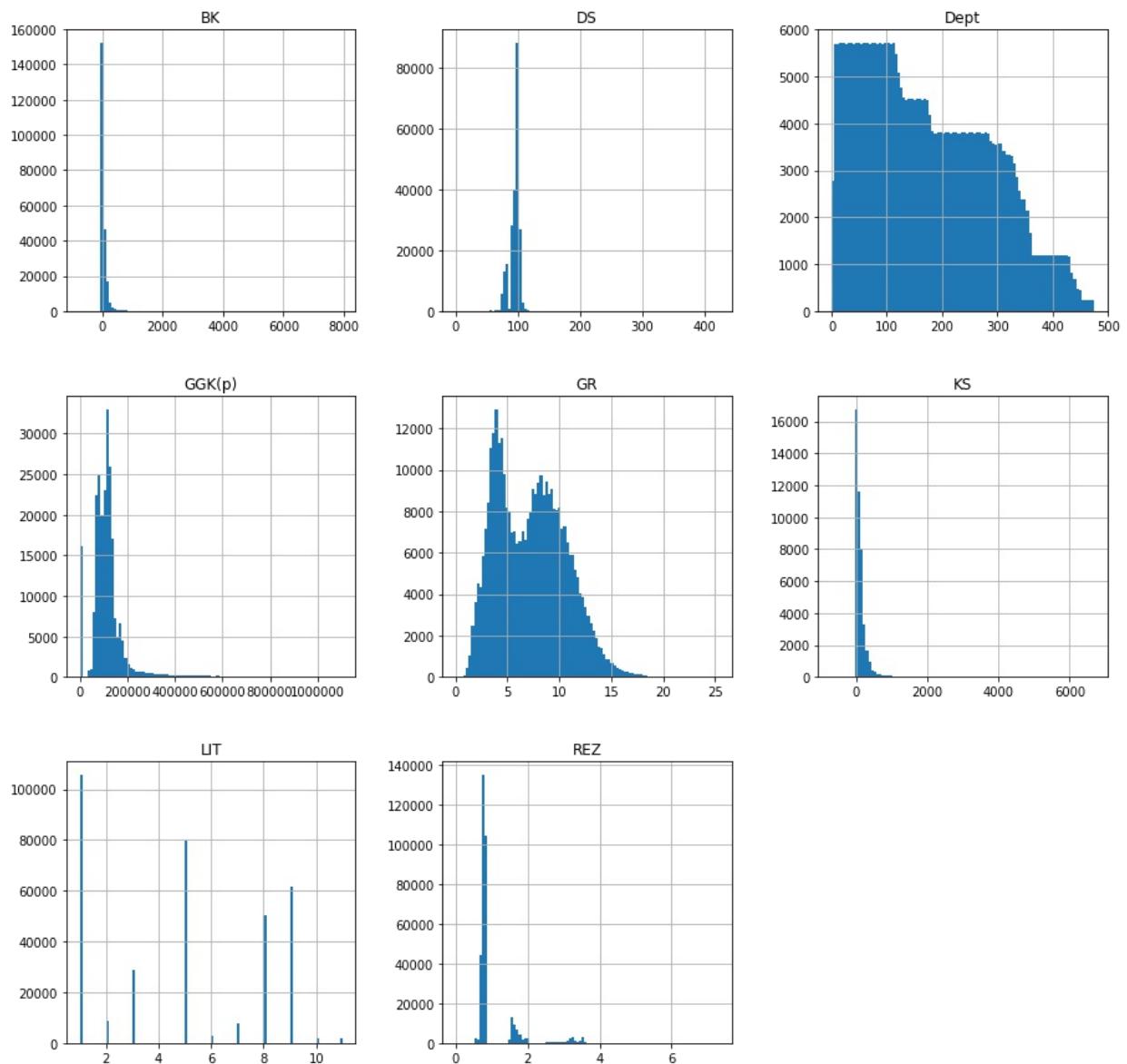


## Распределения переменных

Распределение первенных в первом месторождении

```
dtf_first.hist(bins=100, figsize=(15, 15))
```

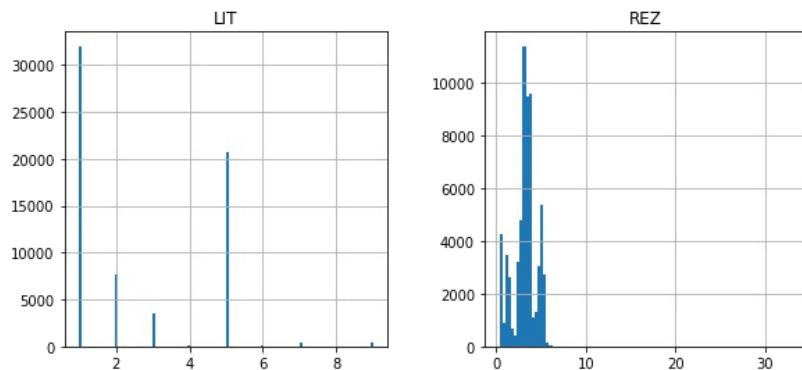
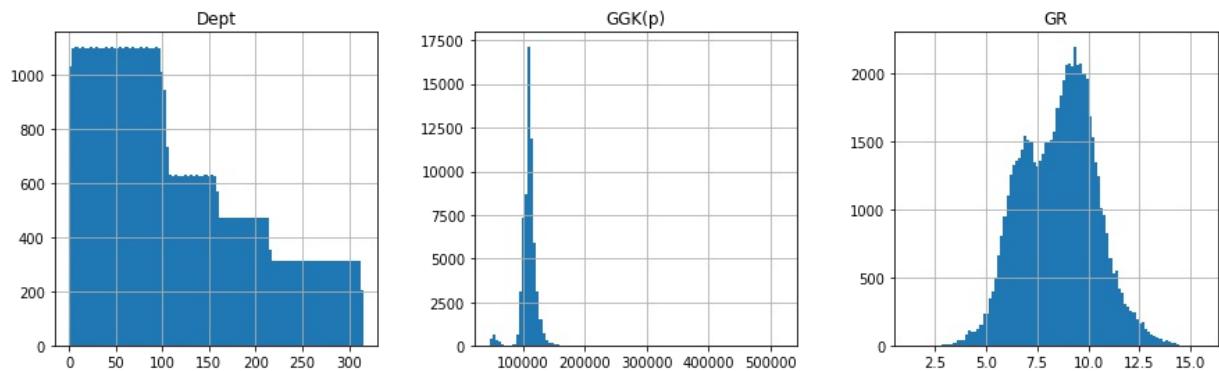
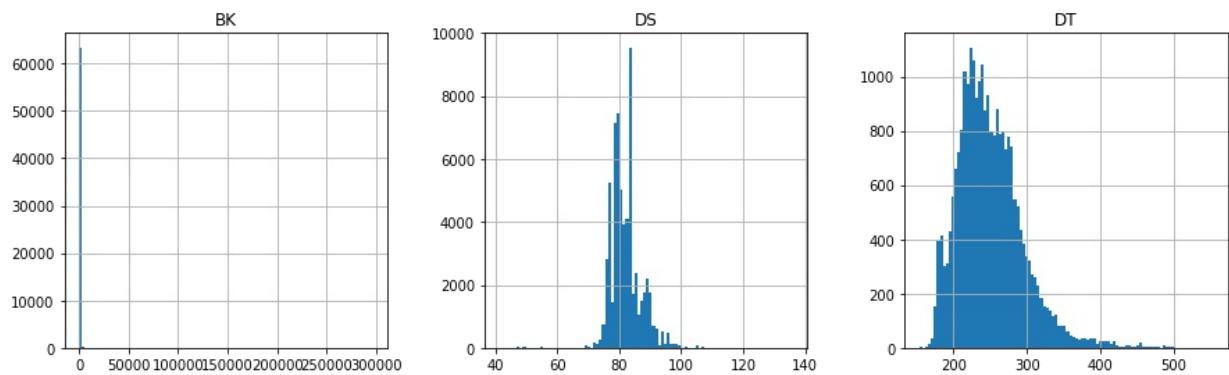
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x12267cc18>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11ee4c240>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x118d924e0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x110efde48>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x118f15d68>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x118f15da0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x111d99ac8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x10d962048>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x115c2f588>]], dtype=object)
```



Распределение первенных во втором месторождении

```
dtf_second.hist(bins=100, figsize=(15, 15))
```

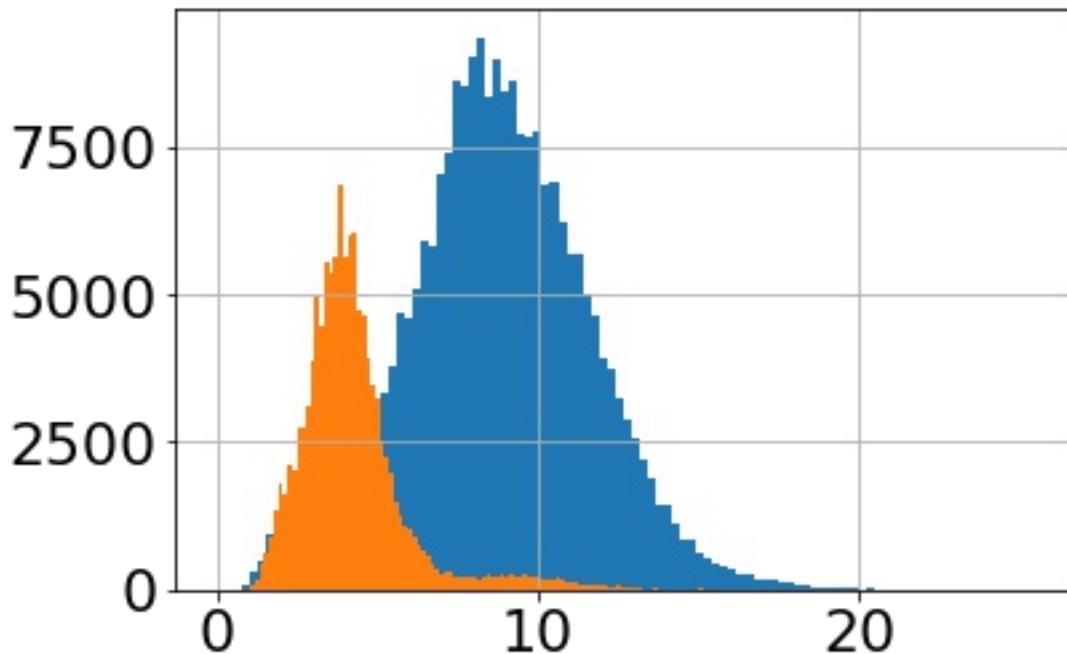
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1147173c8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11abd7f60>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11ac3e860>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x115ccb860>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x115cf8860>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x115cf8898>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x115d902b0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x115dc9780>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11ac51cc0>]], dtype=object)
```



Видно, что у `GK` есть два пика. Первое, что приходит в голову, проверить эмпирику геофизика, что данные в первых 100 метрах не играют роли.

```
dtf_first.GR[dtf_first.Dept >= 100].hist(bins=100)
dtf_first.GR[dtf_first.Dept < 100].hist(bins=100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f569690c3c8>
```

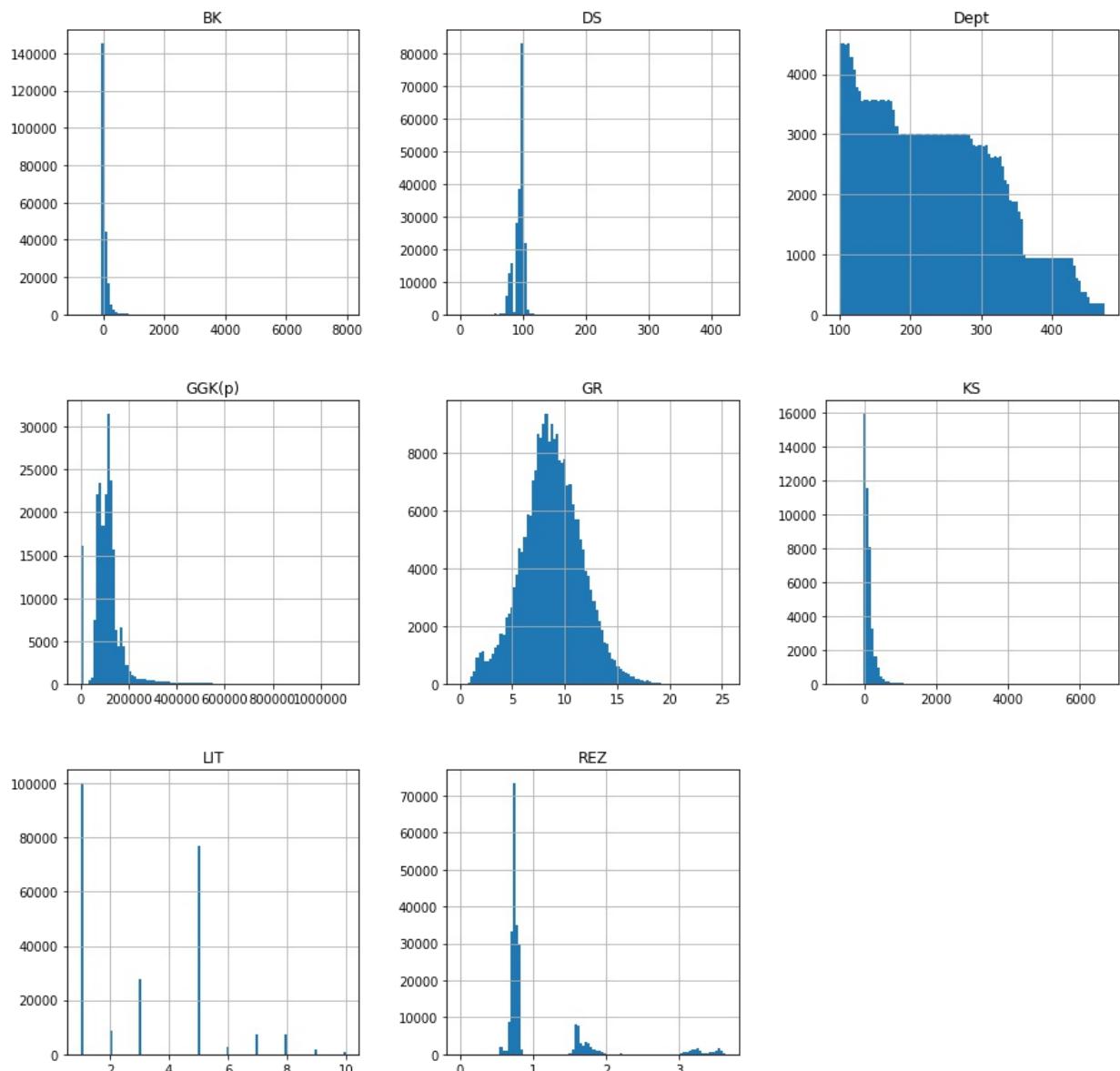


Получается, что действительно, стоит брать данные, которые глубже 100 метров.

Однако обсуждая это явление, вспомнилось, что эти 100 метров есть не всегда, иногда глубже, а иногда вообще нет. Поэтому в конечном приложении имеет смысл дать пользователю выбирать, с какой глубины он хочет начать

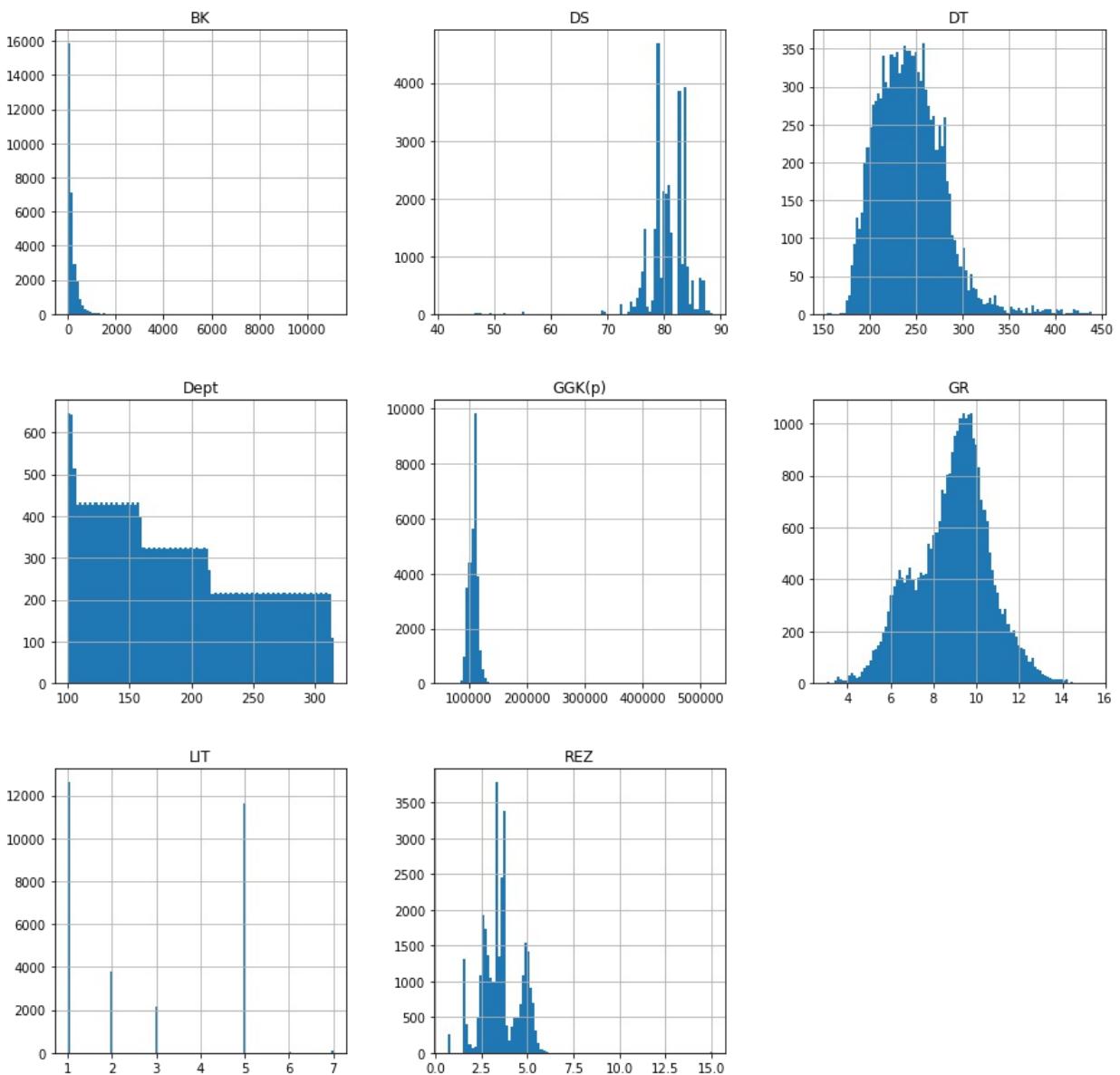
```
dtf_first[dft_first.Dept >= 100].hist(bins=100, figsize=(15, 15))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x123fc9dd8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x124207e48>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x125091e48>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x1250d53c8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x12510f3c8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x12510f400>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x125174828>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1251ae828>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1251ea828>]], dtype=object)
```



```
dtf_second[dtf_second.Dept >= 100].hist(bins=100, figsize=(15, 15))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x12533af28>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11ded3400>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11dee0fd0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x11dee91d0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11dfe8ef0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11dfe8f28>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x11e04a710>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11e024e10>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x11e0aeb70>]], dtype=object)
```



## Анализ гистограмм интервалов относительно значений целевой переменной

```
mpl.rcParams.update({'font.size': 12})
```

```
def plot_histograms_for_target_variables(dtf, save_name=None, figsize=(30, 30), with_names=False):
    """
    function to plot distribution of variables for every value of lithology.
    save_name -- (str) -- file name and where to save plot
    dtf -- datafram with `LIT` column
    with_names -- (bool) -- plot with target variables as numbers or target variable names

    :returns: (Series) value_counts
    """
    # ... (rest of the code)
```

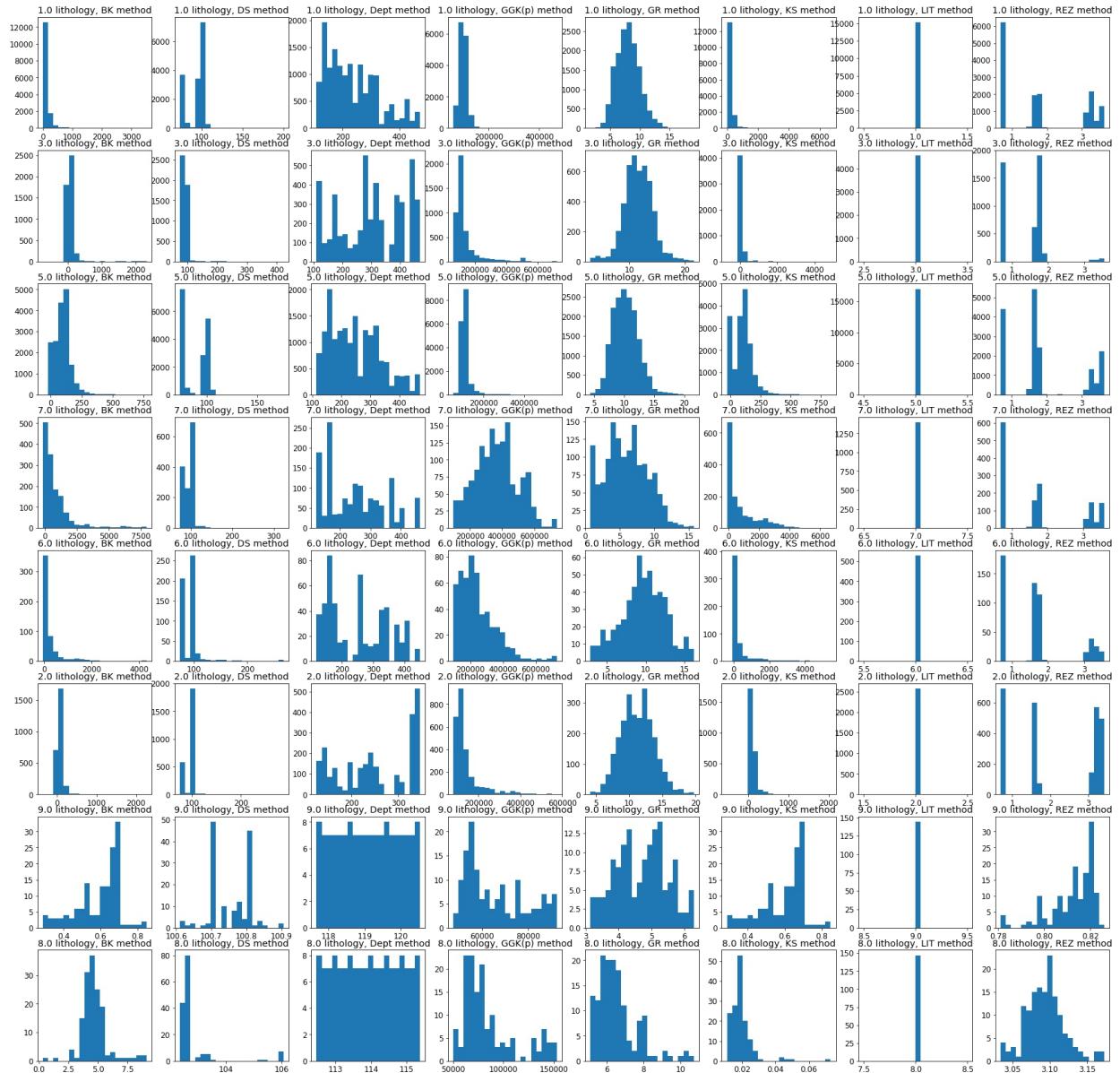
```
counts = dict()

for lit in dtf.dropna().LIT.unique():
    if not pd.isnull(lit):
        descr = dtf.dropna()[dtf.dropna().LIT == lit]
        intervals = dict()
        for v in descr.columns:
            intervals[v] = descr[v].values
        if with_names:
            counts[lithology[lit]] = intervals
        else:
            counts[lit] = intervals
    if with_names:
        fig, axis = plt.subplots(len(counts), len(counts[lithology[1]]), figsize=figsize)
    else:
        fig, axis = plt.subplots(len(counts), len(counts[1]), figsize=figsize)
    for i, val_d in enumerate(counts.items()):
        for j, val in enumerate(val_d[1].items()):
            axis[i, j].hist(val[1], bins=20)
            axis[i, j].set_title('{} lithology, {} method'.format(val_d[0], val[0]))

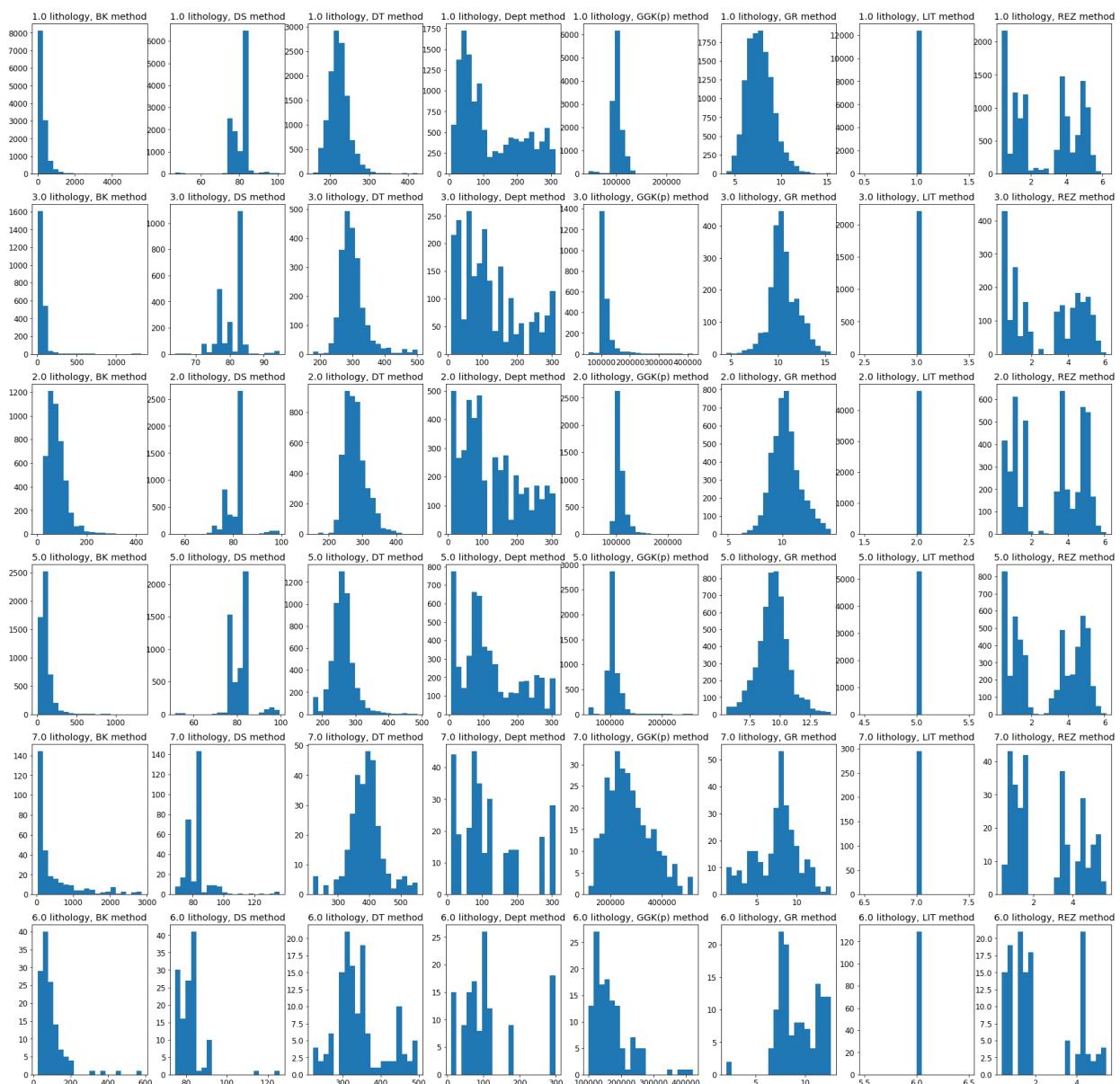
#     plt.tight_layout()
if save_name:
    plt.savefig(save_name)

return counts
```

```
counts = plot_histograms_for_target_variables(dtf_first, save_name='log-images/targ
with_names=False)
```



```
counts = plot_histograms_for_target_variables(dtf_second, save_name='log-images/tar
with_names=False)
```



- GR Подчиняется нормальному распределению
- GGK похожа на нормальное, но имеет тяжелые хвосты, а также выбросы
- KS и BK похожи на распределение Пуассона, за некоторым исключением

В итоге, можно будет попробовать отбрасывать шумовые значения по правилу 2-х СИГМ

## Распределения целевой переменной

```
dtf_first['LIT'].value_counts()
```

1.0	105632
5.0	79680

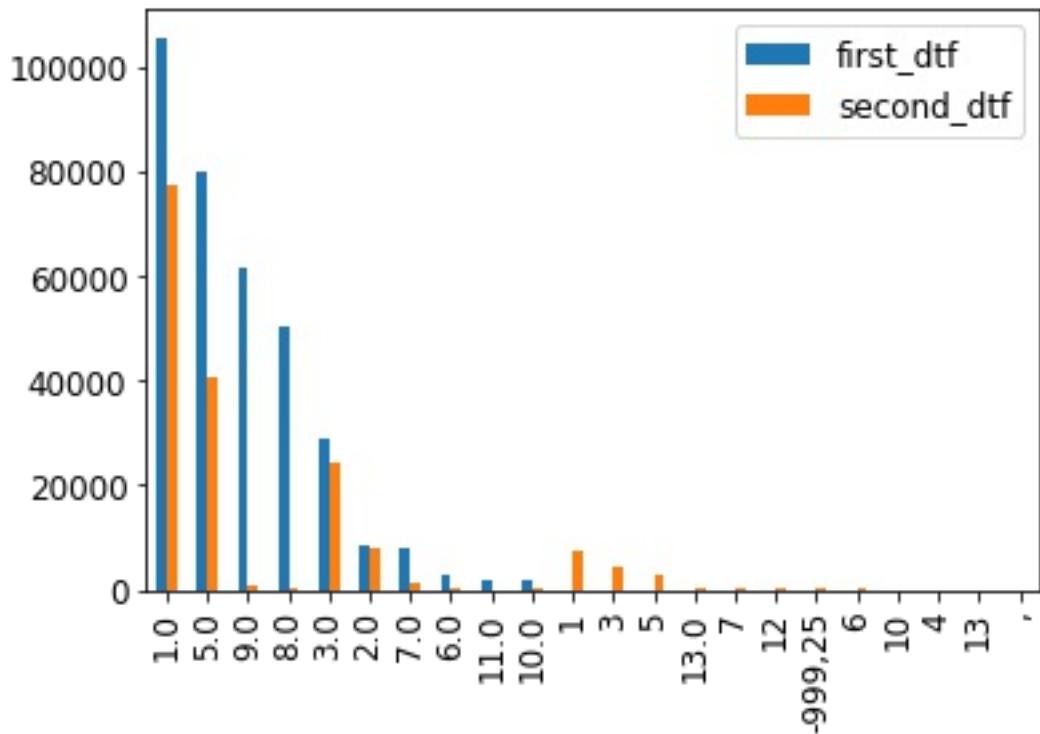
```
9.0      61533
8.0      50142
3.0      28757
2.0      8555
7.0      7821
6.0      2926
11.0     2076
10.0     1969
Name: LIT, dtype: int64
```

```
plt_dtf = pd.DataFrame([dtf_first['LIT'].value_counts(), dtf_second['LIT'].value_counts()])
plt_dtf.columns = ['first_dtf', 'second_dtf']
# plt_dtf.index = map(lambda x: lithology[int(x)], plt_dtf.index)
plt_dtf.plot(kind='bar')
```

```
/home/nikita/projects/carrot/venv/lib/python3.5/site-packages/pandas/core/indexes/api.py:105: FutureWarning: Using a non-unique value as an index is deprecated. Use .set_index(...)[...]
```

```
result = result.union(other)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9fded85438>
```



7 -- Точно уголь

```
plt_dtf.T
```

	песчаник	Аргиллит алевритовый	Аргиллит	Гравелит	алевр
first_dtf	105632.0	8555.0	28757.0	NaN	79680
second_dtf	31961.0	7742.0	3560.0	67.0	20651

## Визуализация первого месторождения с помощью t-SNE

```
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
dtf_first.columns
```

```
Index(['BK', 'DS', 'Dept', 'GGK(p)', 'GR', 'KS', 'LIT', 'REZ'], dtype='object')
```

```
scaler = StandardScaler()
x_data = scaler.fit_transform(dtf_first.dropna()[['BK', 'GGK(p)', 'GR']])
```

```
decomp = TSNE(verbose=0.5)
# decomp = PCA(n_components=2)
decomped_x = decomp.fit_transform(x_data)
```

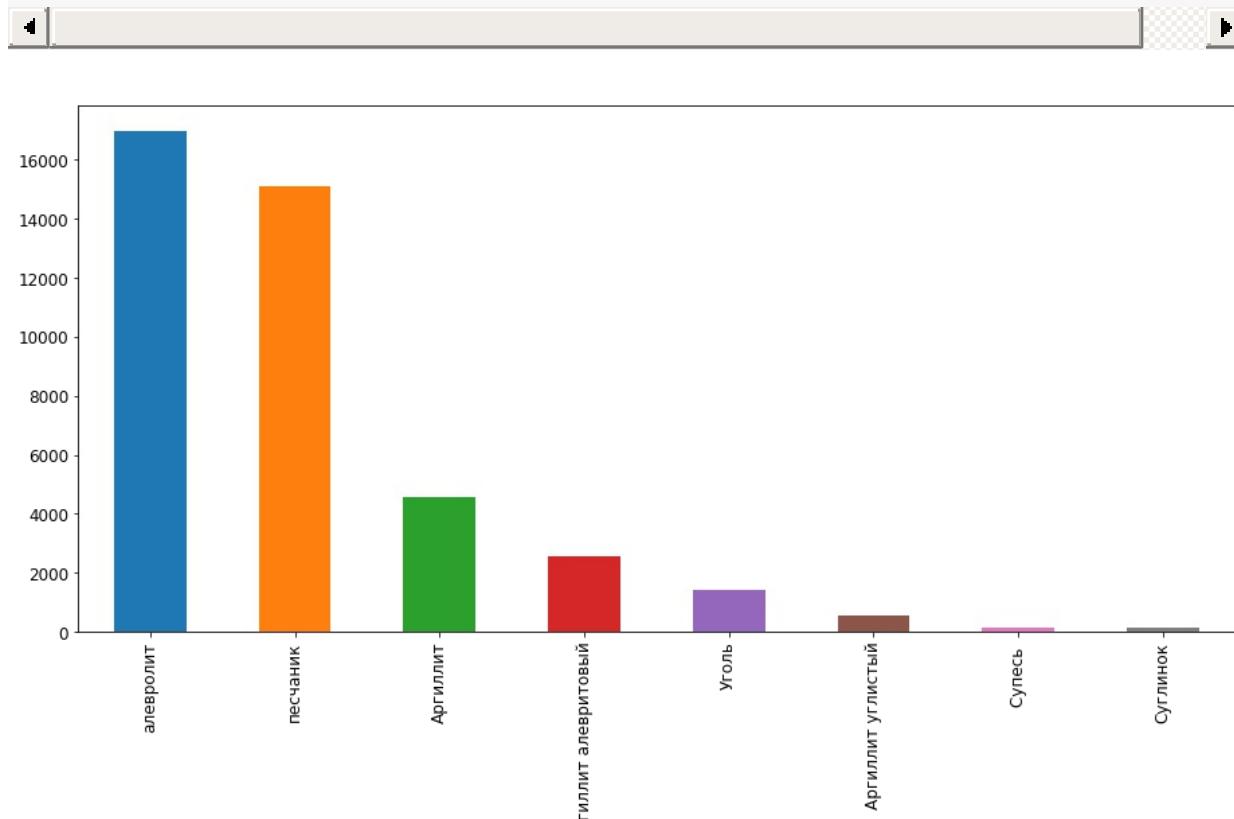
```
[t-SNE] Computing 91 nearest neighbors...
```

```
[t-SNE] Indexed 41447 samples in 0.043s...
[t-SNE] Computed neighbors for 41447 samples in 1.161s...
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.654694
[t-SNE] Error after 1000 iterations: 1.780939
```

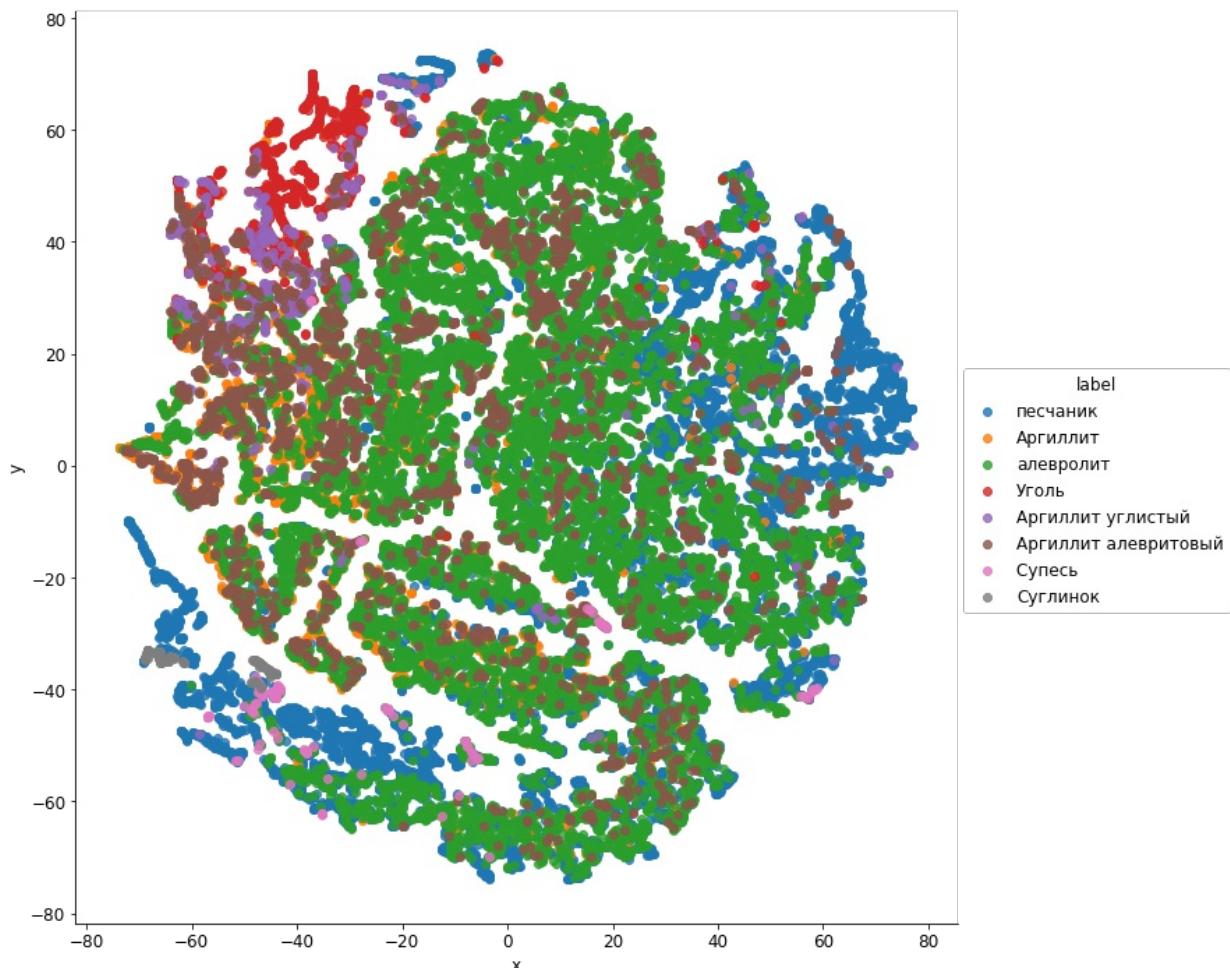
```
dtf_first.dropna().LIT.values, decomped_x
```

```
(array([ 1.,  1.,  1., ...,  1.,  1.,  1.]),
 array([[ -18.70180321,   74.25946045],
        [ 27.37760925,   75.42448425],
        [ 25.98553085,   75.06437683],
        ...,
        [-75.94229126,  -31.71473694],
        [-28.46595573,   11.27650642],
        [ 20.42467499,  -16.33847046]], dtype=float32))
```

```
plt.figure(figsize=(13, 8))
dtf_first.dropna().LIT.apply(lambda x: lithology[x].value_counts()).plot(kind='bar')
plt.tight_layout()
plt.savefig('log-images/first-deposit-target-values.png')
```

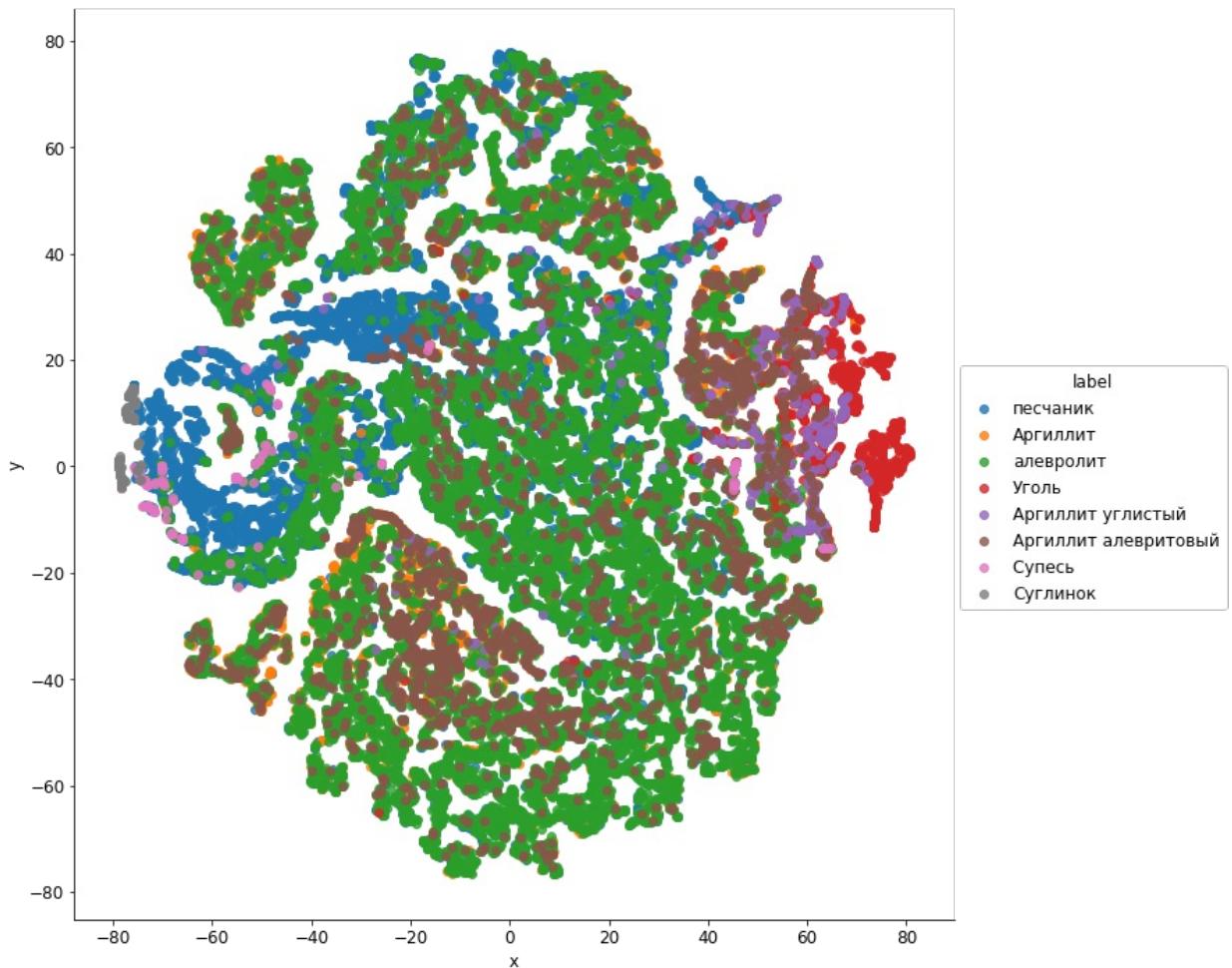


```
sns.lmplot(x='x', y='y', hue='label', data=pd.DataFrame(np.hstack((decomped_x,
                                                               dtf_first.dropna(
                                                               columns=['x', 'y', 'labe
fit_reg=False, size=10)
plt.savefig('log-images/decomposition/decomposition-of-first-deposit-scaled-feats-b
```



```
sns.lmplot(x='x', y='y', hue='label', data=pd.DataFrame(np.hstack((decomped_x,
                                                               dtf_first.dropna(
                                                               columns=['x', 'y', 'labe
fit_reg=False, size=10)
plt.savefig('log-images/decomposition/decomposition-of-first-deposit-scaled-feats-b
```





```

sns.lmplot(x='x', y='y', hue='label', data=pd.DataFrame(np.hstack((decomped_x,
                                                               dtf_first.dropna(
                                                               columns=['x', 'y', 'labe
                                                               fit_reg=False, size=10)
plt.savefig('log-images/decomposition/decomposition-of-secind-deposit-scaled-feats-

```

## Обучение

### Baseline algorithm (count classifier)

Идея: Найти минимальные и максимальные значения для значений методов\*. Затем, на отложенной скважине посчитать попадание значений методов в полученные минимумы и максимумы.

\*Метод -- это метод оценки литологии (Гамма каротаж, ПС, etc.)

```

def count_classifier(train_dtf, with_descr=False):
    """
    Train simple count classifier. Finds (min, max) interval for lithologies
    """

    lit_dict_descr = dict()
    lit_dict = dict()
    for lit in train_dtf.dropna().LIT.unique():
        if not pd.isnull(lit):
            descr = train_dtf.dropna()[train_dtf.dropna().LIT == lit].describe()
            intervals = dict()
            for v in descr.columns:
                intervals[v] = (descr[v]['min'], descr[v]['max'])
            lit_dict_descr[lithology[lit]] = intervals
            lit_dict[lit] = intervals
    if with_descr:
        return pd.DataFrame(lit_dict_descr)
    else:
        return pd.DataFrame(lit_dict)

def in_interval(value, vals):
    """
    check if value between values
    """

    return int(vals[0] <= value <= vals[1])

def predict(clf_dtf, X, threshold=0):
    # Список для сохранения вероятностей
    probs = []
    # Порог для "вероятности"
    threshold = 0
    for row in X.dropna().iterrows():
        dtf_ = pd.DataFrame(columns=[k for k in clf_dtf.keys()],
                            index=X.columns)
        for it, val in row[1].items():
            if it in clf_dtf.index:
                # Проверка, входит ли значение метода в интервал
                mask = clf_dtf.loc[it].apply(lambda x : in_interval(val, x))
                dtf_.loc[it] = mask
        dtf_.dropna(inplace=True)
        # Считаем, для каждой литологии, сколько методов попали в интервал
        prob = dtf_.mean(axis=0)
        # Отсекаем литологии ниже заданного порога
        prob_actual = prob[prob >= threshold]
        probs.append(prob_actual)
    return probs

```

## Разбиение на train/dev/test sets

### Отсутствующие методы

```
methods = ['Dept', 'GGK(p)', 'DS', 'GR', 'BK', 'KS', 'REZ', 'LIT', 'DT']
```

```
for k, dtf in dtfs_third.items():
    tf_list = [str(k)]
    for m in methods:
        if m in dtf.columns:
            tf_list.append('<font size="2" color="green">Yes</font>')
        else:
            tf_list.append('<font size="2" color="red">No</font>')
print('||' + '|'.join(tf_list) + '||')
```

```
|3-100|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>
|3-101|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>
|3-1131-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1133-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1134-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1137-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1138-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1139-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1140-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1141-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1142-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1143-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1146-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1147-d-1|<font size="2" color="green">Yes</font>|<font size="2" color="green">Ye
|3-1147-d-2|<font size="2" color="green">Yes</font>|<font size="2" color="green">Ye
|3-1148-d-1|<font size="2" color="green">Yes</font>|<font size="2" color="green">Ye
|3-1148-d-2|<font size="2" color="green">Yes</font>|<font size="2" color="green">Ye
|3-1159-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1159|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes</f
|3-1167|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes</f
|3-1178|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes</f
|3-1191-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1240-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1240|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes</f
|3-1241-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1244-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1249|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes</f
|3-1304-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1329-d-1|<font size="2" color="green">Yes</font>|<font size="2" color="green">Ye
|3-1329-d-2|<font size="2" color="green">Yes</font>|<font size="2" color="green">Ye
|3-1351-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1365-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1366-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1367-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1375-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-1427|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes</f
|3-1497-d|<font size="2" color="green">Yes</font>|<font size="2" color="green">Yes<
|3-15|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
```





```

|3-33|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-36|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-43|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-44|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-45|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-46|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-47|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-51|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-53|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-57|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-62|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-69|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-88|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|
|3-97|<font size="2" color="green">Yes</font>|<font size="2" color="red">No</font>|

```



Представленные данные имеют такую специфику, что каждая скважина уникальна: В одной скважине могли использоваться одни методы каротажа, в другой другие.

Для понимания ситуации, мы сделали следующую таблицу:

По строкам идут номера скважин, по столбцам, методы и атрибуты скважин.

- Dept -- глубина
- GGK(p) -- Гамма-гамма каротаж
- DS -- Диаметр скважины
- GR -- Гамма каротаж
- DK -- Боковой каротаж
- KS -- Кажущееся сопротивление
- REZ -- Электрическое сопротивление в скважине (или вроде того)
- LIT -- Номер литологии
- DT -- Акустический каротаж

well/method	Dept	GGK(p)	DS	GR	BK	KS	REZ
1787	Yes	Yes	Yes	Yes	Yes	No	Yes
628	Yes	Yes	Yes	Yes	Yes	No	Yes
1790-1	Yes	Yes	Yes	Yes	Yes	No	Yes
1399	Yes	Yes	Yes	Yes	Yes	No	Yes
690	Yes	Yes	Yes	Yes	Yes	No	Yes

1731	Yes	Yes	Yes	Yes	Yes	No	Yes
1790-3	Yes	Yes	Yes	Yes	Yes	No	Yes
1790-4	Yes	Yes	Yes	Yes	Yes	No	Yes
1801	Yes	Yes	Yes	Yes	Yes	No	Yes
1093	Yes						
210	Yes						
788	Yes	Yes	Yes	Yes	Yes	No	Yes
1804	Yes	Yes	Yes	Yes	Yes	No	Yes
1779	Yes	Yes	Yes	Yes	Yes	No	Yes
1462	Yes						
809	Yes	Yes	Yes	Yes	Yes	No	Yes
639	Yes	Yes	Yes	Yes	Yes	No	Yes
875	Yes	Yes	No	Yes	No	Yes	Yes
1763	Yes						
1742	Yes	Yes	Yes	Yes	Yes	No	Yes
822	Yes	Yes	Yes	Yes	Yes	No	Yes
1790-2	Yes	Yes	Yes	Yes	Yes	No	Yes
919	Yes	Yes	Yes	Yes	Yes	No	Yes
1725	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-751	Yes	No	Yes	Yes	No	Yes	No
UK-754	Yes	No	Yes	Yes	No	Yes	No
UK-745	Yes	No	Yes	Yes	No	Yes	No

UK-741	Yes	No	Yes	Yes	No	Yes	No
UK-743	Yes	No	Yes	Yes	No	Yes	No
UK-742	Yes	No	Yes	Yes	No	Yes	No
UK-752	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-758	Yes	No	Yes	Yes	No	Yes	No
UK-740	Yes	No	Yes	Yes	No	Yes	No
UK-762	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-746-1	Yes	No	Yes	Yes	No	Yes	No
UK-739	Yes	No	Yes	Yes	No	Yes	No
UK-749	Yes	No	Yes	Yes	No	Yes	No
UK-757-1	Yes	No	Yes	Yes	No	Yes	No
UK-753	Yes	No	Yes	Yes	No	Yes	No
UK-752-1	Yes	No	Yes	Yes	No	Yes	No
UK-757	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-744	Yes	No	Yes	Yes	No	Yes	No
UK-756	Yes	No	Yes	Yes	No	Yes	No
UK-750	Yes	No	Yes	Yes	No	Yes	No
UK-761	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-738	Yes	No	Yes	Yes	No	Yes	No
UK-760	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-746	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-748	Yes	No	Yes	Yes	No	Yes	No

UK-759	Yes	Yes	Yes	Yes	Yes	No	Yes
UK-755	Yes	No	Yes	Yes	No	Yes	No
UK-747	Yes	No	Yes	Yes	No	Yes	No
3-100	Yes	No	No	No	No	Yes	No
3-101	Yes	No	No	No	No	Yes	No
3-1131-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1133-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1134-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1137-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1138-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1139-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1140-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1141-d	Yes	Yes	No	Yes	No	No	No
3-1142-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1143-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1146-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1147-d-1	Yes	Yes	Yes	Yes	No	Yes	No
3-1147-d-2	Yes	Yes	Yes	Yes	No	Yes	No
3-1148-d-1	Yes	Yes	Yes	Yes	No	Yes	No
3-1148-d-2	Yes	Yes	Yes	Yes	No	Yes	No
3-1159-d	Yes	Yes	No	Yes	No	Yes	No

3-1159	Yes	Yes	Yes	Yes	No	Yes	No
3-1167	Yes	Yes	Yes	Yes	No	Yes	No
3-1178	Yes	Yes	Yes	Yes	No	Yes	No
3-1191-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1240-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1240	Yes	Yes	Yes	Yes	No	Yes	No
3-1241-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1244-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1249	Yes	Yes	Yes	Yes	No	Yes	No
3-1304-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1329-d-1	Yes	Yes	Yes	Yes	No	Yes	No
3-1329-d-2	Yes	Yes	Yes	Yes	No	Yes	No
3-1351-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1365-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1366-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1367-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1375-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1427	Yes	Yes	Yes	Yes	No	Yes	No
3-1497-d	Yes	Yes	Yes	Yes	No	Yes	No
3-15	Yes	No	No	No	No	Yes	No
3-156-d	Yes	No	No	No	No	Yes	No
3-156							

	Yes	No	No	No	No	Yes	No
3-159-d	Yes	No	No	No	No	Yes	No
3-159	Yes	No	No	No	No	Yes	No
3-160	Yes	No	No	No	No	Yes	No
3-1662-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1668-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1669-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1684-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1688-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1689-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1690-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1691-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1692-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1694-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1698-d	Yes	Yes	Yes	Yes	No	Yes	No
3-17	Yes	No	No	No	No	Yes	No
3-170-d	Yes	No	No	No	No	Yes	No
3-1706-d	Yes	Yes	Yes	Yes	No	No	No
3-1707-d	Yes	Yes	Yes	Yes	No	Yes	No
3-171-d	Yes	No	No	No	No	Yes	No
3-1711-d	Yes	Yes	Yes	Yes	No	Yes	No

3-1712-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1713-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1714-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1715-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1716-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1717-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1719-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1720-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1721-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1722-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1723-d	Yes	Yes	Yes	Yes	No	Yes	No
3-173-d	Yes	No	No	No	No	Yes	No
3-173	Yes	No	No	No	No	Yes	No
3-1733-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1734-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1739-d	Yes	Yes	Yes	Yes	No	Yes	No
3-174-d	Yes	No	No	No	No	Yes	No
3-174	Yes	No	No	No	No	Yes	No
3-1740-d	Yes	Yes	Yes	Yes	No	Yes	No
3-1747-d	Yes	Yes	Yes	Yes	No	Yes	No
3-19	Yes	No	No	No	No	Yes	No
3-191	Yes	No	No	No	No	Yes	No

3-192	Yes	No	No	No	No	Yes	No
3-21	Yes	No	No	No	No	Yes	No
3-219	Yes	No	No	No	No	Yes	No
3-220-1	Yes	No	No	No	No	Yes	No
3-220	Yes	No	No	No	No	Yes	No
3-221	Yes	No	No	No	No	Yes	No
3-222	Yes	No	No	No	No	Yes	No
3-223	Yes	No	No	No	No	Yes	No
3-224	Yes	No	No	No	No	Yes	No
3-225	Yes	No	No	No	No	Yes	No
3-226	Yes	No	No	No	No	Yes	No
3-231	Yes	No	No	No	No	Yes	No
3-232	Yes	No	No	No	No	Yes	No
3-233	Yes	No	No	No	No	Yes	No
3-234	Yes	No	No	No	No	Yes	No
3-235	Yes	No	No	No	No	Yes	No
3-236	Yes	No	No	No	No	Yes	No
3-237	Yes	No	No	No	No	Yes	No
3-238	Yes	No	No	No	No	Yes	No
3-239	Yes	No	No	No	No	Yes	No
3-240	Yes	No	No	No	No	Yes	No
3-241	Yes	No	No	No	No	Yes	No

3-242	Yes	No	No	No	No	Yes	No
3-243	Yes	No	No	No	No	Yes	No
3-244	Yes	No	No	No	No	Yes	No
3-245	Yes	No	No	No	No	Yes	No
3-246	Yes	No	No	No	No	Yes	No
3-247	Yes	No	No	No	No	Yes	No
3-248	Yes	No	No	No	No	Yes	No
3-249	Yes	No	No	No	No	Yes	No
3-250	Yes	No	No	No	No	Yes	No
3-252	Yes	No	No	No	No	Yes	No
3-256	Yes	No	No	No	No	Yes	No
3-257	Yes	No	No	No	No	Yes	No
3-259	Yes	No	No	No	No	Yes	No
3-26	Yes	No	No	No	No	Yes	No
3-260	Yes	No	No	No	No	Yes	No
3-2605	Yes	No	Yes	Yes	No	Yes	Yes
3-2613	Yes	No	Yes	Yes	No	Yes	Yes
3-2614	Yes	No	Yes	Yes	No	Yes	Yes
3-2615-d	Yes	No	Yes	Yes	No	Yes	Yes
3-2615	Yes	No	Yes	Yes	No	Yes	Yes
3-2616-d	Yes	No	Yes	Yes	No	Yes	Yes
3-2617-d	Yes	No	Yes	Yes	No	Yes	No

3-2618-d	Yes	No	Yes	Yes	No	Yes	Yes
3-2618	Yes	No	Yes	Yes	No	Yes	Yes
3-2619	Yes	No	Yes	Yes	No	Yes	Yes
3-2620-d	Yes	No	No	Yes	No	Yes	No
3-2620	Yes	No	Yes	Yes	No	Yes	Yes
3-2621-d	Yes	No	Yes	Yes	No	Yes	Yes
3-2621	Yes	No	Yes	Yes	No	Yes	Yes
3-2622	Yes	No	Yes	Yes	No	Yes	Yes
3-2623-d	Yes	No	Yes	Yes	No	Yes	Yes
3-2623	Yes	No	Yes	Yes	No	Yes	Yes
3-2624	Yes	No	Yes	Yes	No	Yes	Yes
3-2625-d	Yes	No	Yes	Yes	No	Yes	No
3-2626	Yes	No	Yes	Yes	No	Yes	Yes
3-2627	Yes	No	Yes	Yes	No	Yes	Yes
3-2628	Yes	No	Yes	Yes	No	Yes	Yes
3-2629	Yes	No	Yes	Yes	No	Yes	Yes
3-2630	Yes	No	Yes	Yes	No	Yes	Yes
3-2631	Yes	No	Yes	Yes	No	Yes	Yes
3-2632	Yes	No	Yes	Yes	No	Yes	Yes
3-2633	Yes	No	Yes	Yes	No	Yes	Yes
3-2634	Yes	No	Yes	Yes	No	Yes	Yes
3-265	Yes	No	No	No	No	Yes	No

3-269	Yes	No	No	No	No	Yes	No
3-29	Yes	No	No	No	No	Yes	No
3-33	Yes	No	No	No	No	Yes	No
3-36	Yes	No	No	No	No	Yes	No
3-43	Yes	No	No	No	No	Yes	No
3-44	Yes	No	No	No	No	Yes	No
3-45	Yes	No	No	No	No	Yes	No
3-46	Yes	No	No	No	No	Yes	No
3-47	Yes	No	No	No	No	Yes	No
3-51	Yes	No	No	No	No	Yes	No
3-53	Yes	No	No	No	No	Yes	No
3-57	Yes	No	No	No	No	Yes	No
3-62	Yes	No	No	No	No	Yes	No
3-69	Yes	No	No	No	No	Yes	No
3-88	Yes	No	No	No	No	Yes	No
3-97	Yes	No	No	No	No	Yes	No

Как видно в таблице, есть методы, которые всегда используются:

- [GGK, GR, REZ], также, за исключением одного случая, используется BK.
- DT Используется только в одном месторождении и то не во всех случаях (что странно, на самом деле)
- KS Также используется, но далеко не во всех скважинах.

Поскольку количество фичей в модели, которую мы будем обучать, фиксировано, то имеет смысл рассмотреть аппроксимацию отсутствующих методов, поскольку у нас есть два варианта работы с отсутствующими фичами:

- Вообще не рассматривать их при построении модели
- Заполнить отсутствующие значения каким-нибудь методом
  - Средним по скважинам
  - "Нулевым" значением
  - Апроксимировать значение

## Выборка без отсутствующих методов

Пусть одна скважина в первом месторождении будет тестовая, а второе месторождение будет валидационным

```
seed = 30
```

```
X_data = ['GR', 'BK', 'GGK(p)', 'DS', 'REZ', 'Dept']
y_data = ['LIT']
```

```
excluded_dts = ['875']
np.random.seed(seed)
test_dtf_number = np.random.choice([k for k in dts.keys() if k != excluded_dts[0]])
excluded_dts.append(test_dtf_number)
print(excluded_dts[1])
```

```
1742
```

```
train_data = pd.concat([d for k, d in dts.items() if k not in excluded_dts])
test_data = pd.concat([d for k, d in dts.items() if k not in excluded_dts])
train_data[X_data + y_data].dropna().shape, test_data[X_data + y_data].dropna().shape
```

```
((199256, 7), (62782, 7))
```

```
# Берем данные глубже 100 метров, потому что см. анализ данных
x_train = train_data[train_data.Dept > 100][X_data + y_data].dropna()[X_data]
```

```
x_train.drop('Dept', inplace=True, axis=1)
y_train = train_data[train_data.Dept > 100][X_data + y_data].dropna()[y_data]
x_train.shape, y_train.shape
```

```
((193866, 5), (193866, 1))
```

```
x_test = test_data[test_data.Dept > 100][X_data + y_data].dropna()[X_data]
x_test.drop('Dept', inplace=True, axis=1)
y_test = test_data[test_data.Dept > 100][X_data + y_data].dropna()[y_data]
x_test.shape, y_test.shape
```

```
((30119, 5), (30119, 1))
```

## Обучение методов

### Random Forest

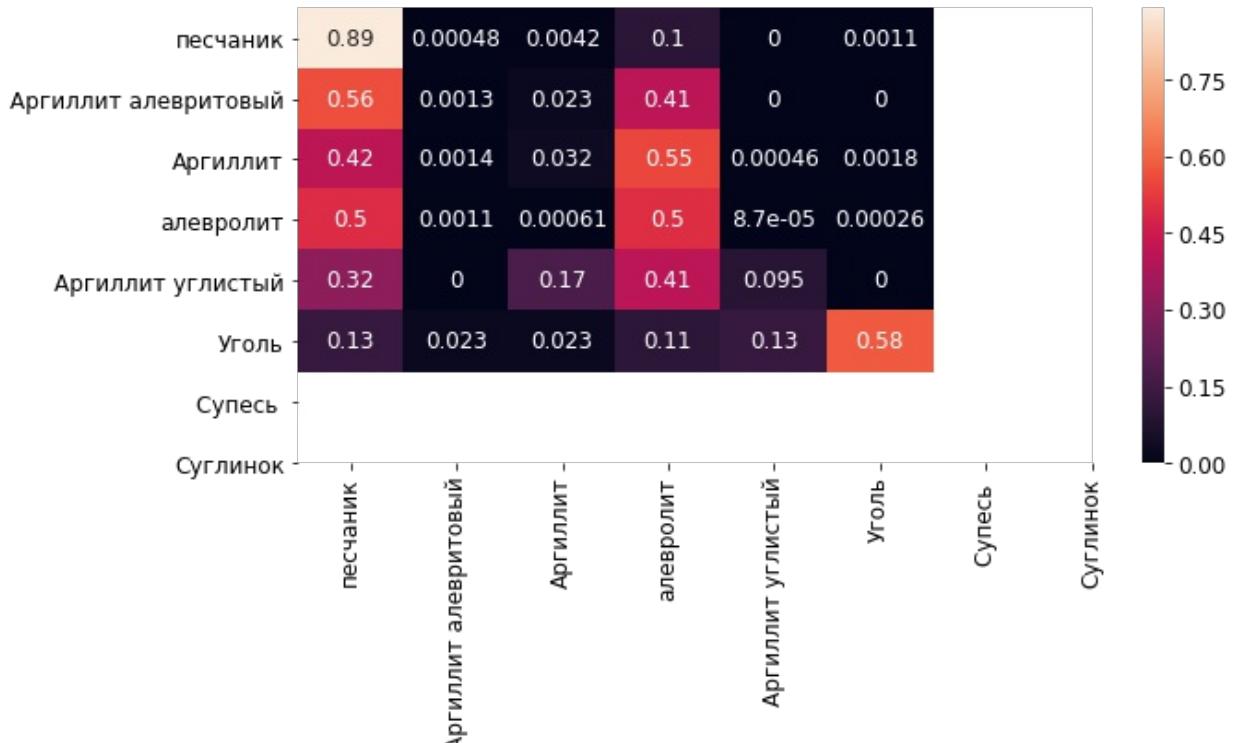
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

```
params = {'n_estimators':10,
          'max_depth':10,
          'class_weight':'balanced',
          'random_state' : seed}
clf = RandomForestClassifier(**params)
clf.fit(x_train, y_train)
clf.score(x_train, y_train), clf.score(x_test, y_test)
```

```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python
```

```
(0.65721684049807594, 0.56648627112453931)
```

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]
plt.figure(figsize=(10, 6))
plot_conusion_matrix(confusion_matrix(y_test, clf.predict(x_test)), ticks=ticks, sa
    file_name='RandomForest_' + '_'.join(['{}-{}'.format(k, v)for
```



## LogisticRegression

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

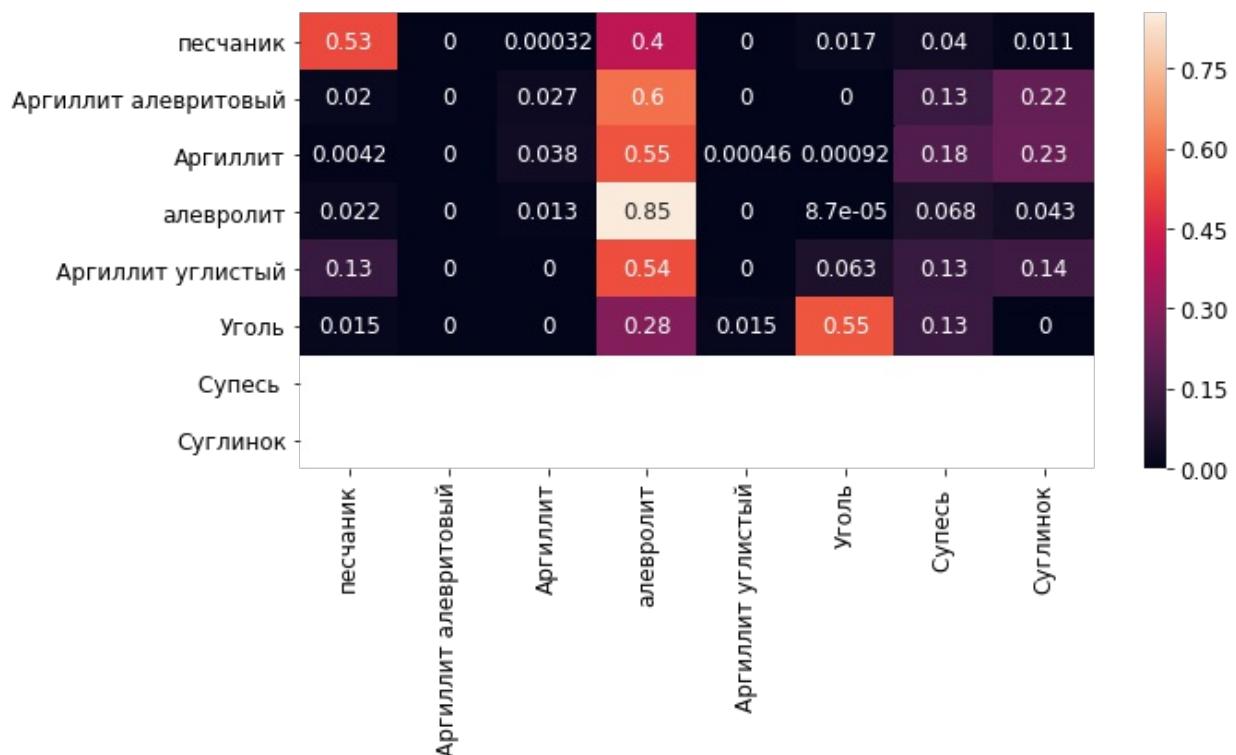
```
params = {'class_weight': 'balanced',
          'random_state' : seed}
clf = LogisticRegression(**params)
clf.fit(x_train_scaled, y_train)
clf.score(x_train_scaled, y_train), clf.score(x_test_scaled, y_test)
```

```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python  
y = column_or_1d(y, warn=True)
```

```
(0.57067252638420352, 0.5512467213386899)
```

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]  
plt.figure(figsize=(10, 6))  
plot_conusion_matrix(confusion_matrix(y_test, clf.predict(x_test_scaled)), ticks=ticks,  
file_name='LogisticRegression_' + '_'.join(['{}-{}'.format(k,
```

```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python
```



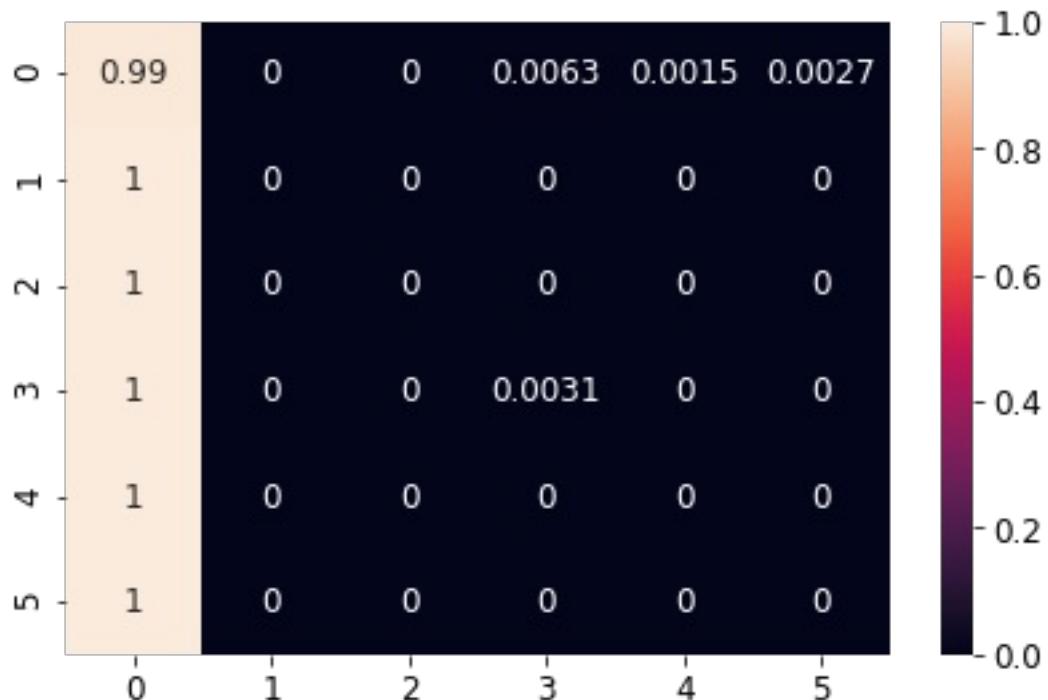
## Count classifier

```
x_train['LIT'] = y_train
clf_dtf = count_classifier(x_train)
probs = predict(pd.DataFrame(clf_dtf).loc[X_data[:,-1]], x_test)
x_train.drop('LIT', inplace=True, axis=1)
```

```
threshold = 5/6
count = 0
total = 0
Y_true = []
Y_pred = []
for y_true_val, y_pred in zip(y_test.values, probs):
    total += 1
    if len(y_pred) > 0:
        count += (y_true_val in y_pred[y_pred > threshold].values)
        Y_pred.append(y_pred.argmax())
    else:
        Y_pred.append(-1)
Y_true.append(y_true_val)
print(count/total)
```

```
0.1870580032537601
```

```
plot_conusion_matrix(confusion_matrix(Y_true, Y_pred), save_fig=False)
```



## **Выводы:**

Результаты классификации с помощью Random Forest и Logistic Regression показывают, что необходимо использовать понижение количества наблюдаемых занчений (Under sampling)

Стоит добавить:

- Кросс валидацию с one deposit out;
- Under sampling
- Попробовать альтернативные методы классификаци

Count classifier показывает не очень впечатляющие результаты. Возможные причины проблемы:

- Шумовые значения
- Использование лишних данных (DS, REZ не столь важны, как GK или GGK)

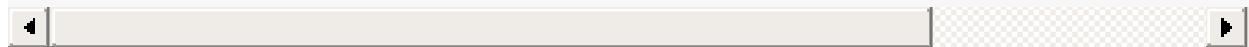
## **Заполнить пропущенные методы и значения средним по датасету**

```
seed = 30
```

```
X_data = ['GR', 'BK', 'KS', 'GGK(p)', 'DS', 'REZ', 'Dept']
X_test_data = ['GR', 'GGK(p)', 'DS', 'REZ', 'Dept']
y_data = ['LIT']
```

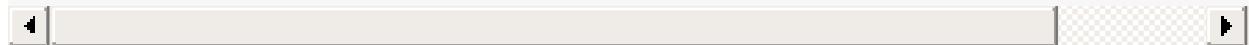
```
excluded_dts = [875]
np.random.seed(seed)
test_dtf_number = np.random.choice([k for k in dts.keys() if k not in excluded_dts])
excluded_dts.append(test_dtf_number)
print(excluded_dts[1])
```

```
train_data = pd.concat([d for k, d in dtfs_first.items() if k not in excluded_dtfs]
test_data = pd.concat([d for k, d in dtfs_second.items() if k not in excluded_dtfs]
train_data[X_data].shape, test_data[X_test_data].shape
```



```
((331632, 7), (65457, 5))
```

```
# Берем данные глубже 100 метров, потому что см. анализ данных
x_train = train_data.iloc[train_data[train_data.Dept > 100][y_data].dropna().index]
x_train.drop('Dept', inplace=True, axis=1)
y_train = train_data.iloc[train_data[train_data.Dept > 100][y_data].dropna().index]
x_train.shape, y_train.shape
```



```
((222724, 6), (222724, 1))
```

```
x_test = test_data.iloc[test_data[test_data.Dept > 100][y_data].dropna().index][X_t
x_test.drop('Dept', inplace=True, axis=1)
y_test = test_data.iloc[test_data[test_data.Dept > 100][y_data].dropna().index][y_d
x_test.shape, y_test.shape
```



```
((30368, 4), (30368, 1))
```

```
fill_values = x_train.describe().loc['mean']
x_train.fillna(fill_values, inplace=True, axis=0)
print()
```

```
x_test['KS'] = np.nan
x_test['BK'] = np.nan
x_test.fillna(fill_values, inplace=True, axis=0)
print()
```

## Обучение методов

### Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

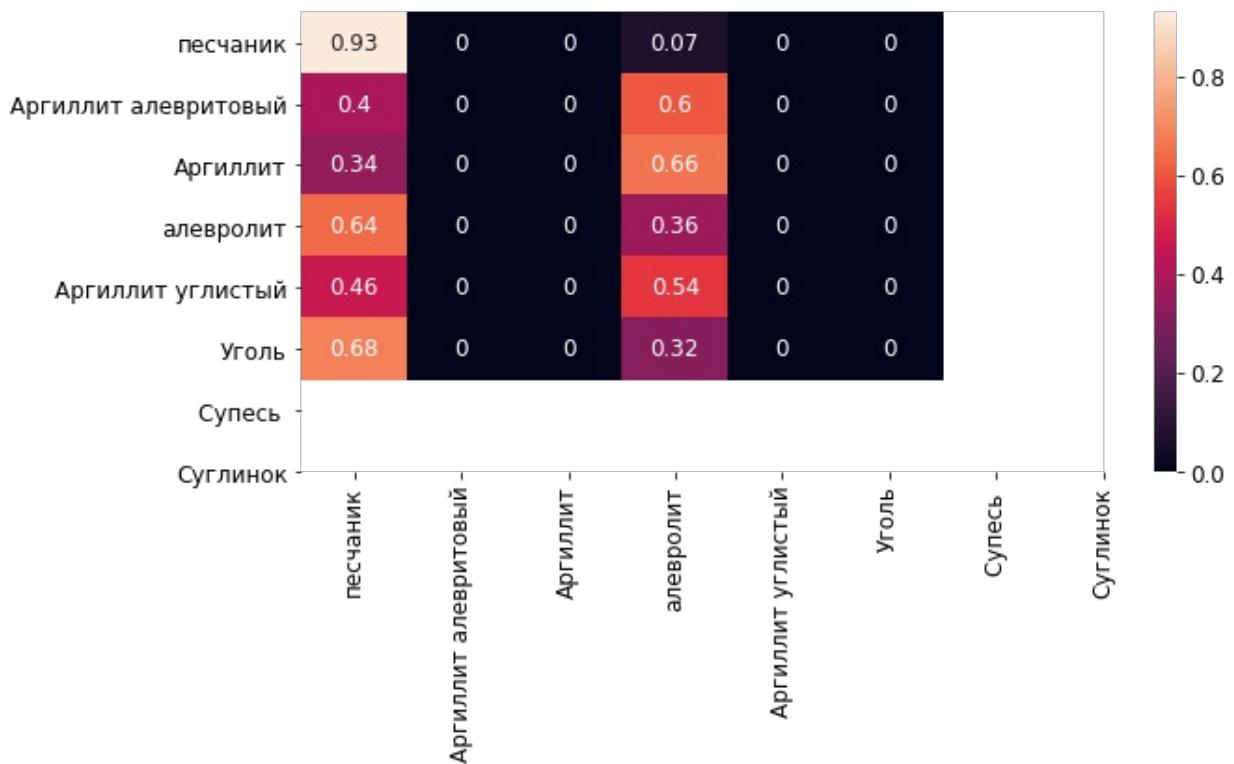
```
params = {'n_estimators':10,
          'max_depth':12,
          'class_weight':'balanced',
          'random_state' : seed}
clf = RandomForestClassifier(**params)
clf.fit(x_train, y_train)
clf.score(x_train, y_train), clf.score(x_test, y_test)
```

```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python
```

```
(0.7045581077926043, 0.52433482613277138)
```

Забавная зависимость: Если брать максимальную глубину 10, то получается очень сильный underfit (0.04), а если брать 12, то точность возрастает до 0.52

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]
plt.figure(figsize=(10, 6))
plot_conusion_matrix(confusion_matrix(y_test, clf.predict(x_test)), ticks=ticks, sa
file_name='RandomForest_' + '_'.join(['{}-{}'.format(k, v)for
```



Заполнение средними значениями дает не очень хорошие результаты. Скорее всего, дело в том, что:

1. на разной глубине разные значения методов;
2. Мы добавляем еще больше больше лишних значений (в частности песка), поэтому разумно было бы использовать undersampling

## LogisticRegression

```
from sklearn.preprocessing import StandardScaler
```

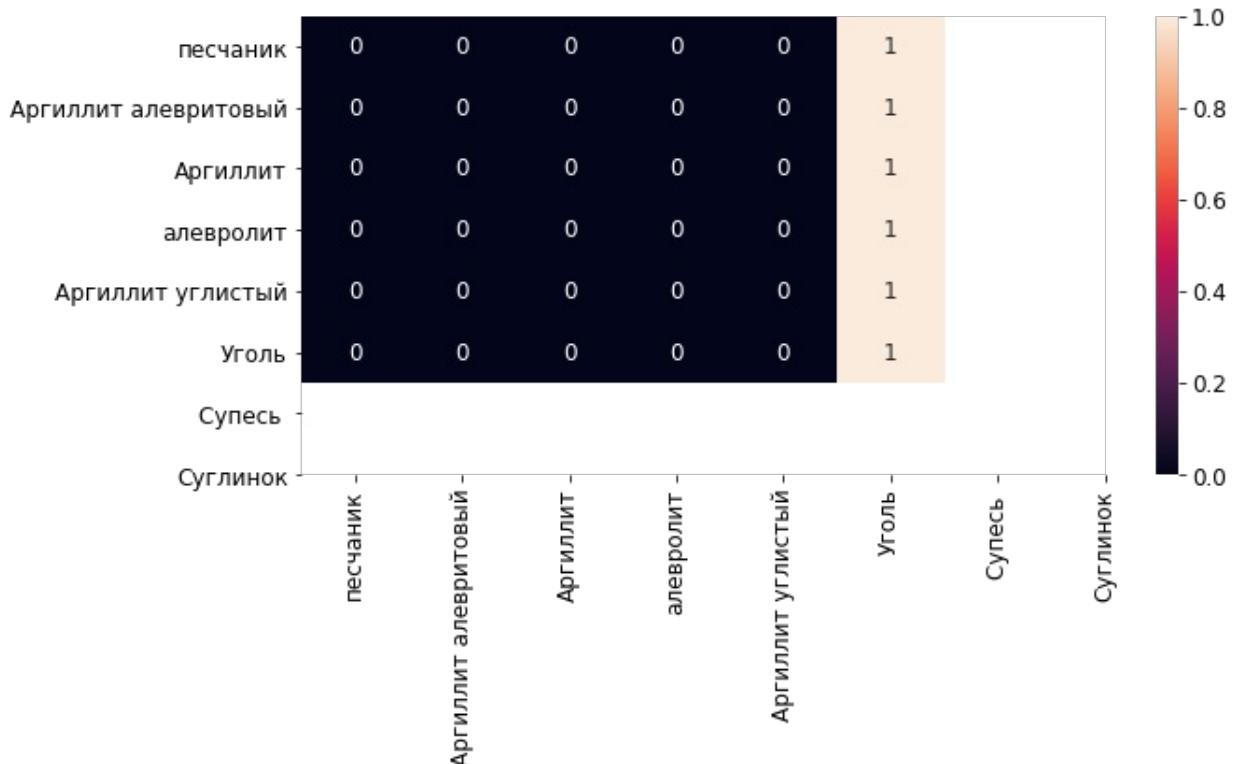
```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
params = {'class_weight': 'balanced',
          'random_state' : seed}
clf = LogisticRegression(**params)
clf.fit(x_train_scaled, y_train)
clf.score(x_train_scaled, y_train), clf.score(x_test_scaled, y_test)
```

```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python  
y = column_or_1d(y, warn=True)
```

```
(0.54122591189095026, 0.0042808219178082189)
```

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]  
plt.figure(figsize=(10, 6))  
plot_confusion_matrix(confusion_matrix(y_test, clf.predict(x_test_scaled)), ticks=ticks,  
file_name='LogisticRegression_' + '_'.join(['{}-{}'.format(k,
```



Логистическая регрессия также показывает плохие результаты :(

**Заполнить пропущенные методы и значения "нулями"**

```
seed = 30
```

```
X_data = ['GR', 'BK', 'KS', 'GGK(p)', 'DS', 'REZ', 'Dept']
X_test_data = ['GR', 'GGK(p)', 'DS', 'REZ', 'Dept']
y_data = ['LIT']
```

```
excluded_dtsfs = ['875']
np.random.seed(seed)
test_dtf_number = np.random.choice([k for k in dtsfs_first.keys() if k not in excluded_dtsfs])
excluded_dtsfs.append(test_dtf_number)
print(excluded_dtsfs[1])
```

```
1742
```

```
train_data = pd.concat([d for k, d in dtsfs_first.items() if k not in excluded_dtsfs]
test_data = pd.concat([d for k, d in dtsfs_second.items() if k not in excluded_dtsfs])
train_data[X_data].shape, test_data[X_test_data].shape
```

```
((331632, 7), (65457, 5))
```

```
# Берем данные глубже 100 метров, потому что см. анализ данных
x_train = train_data.iloc[train_data[train_data.Dept > 100][y_data].dropna().index]
x_train.drop('Dept', inplace=True, axis=1)
y_train = train_data.iloc[train_data[train_data.Dept > 100][y_data].dropna().index]
x_train.shape, y_train.shape
```

```
((222724, 6), (222724, 1))
```

```
x_test = test_data.iloc[test_data[test_data.Dept > 100][y_data].dropna().index][X_test_data]
x_test.drop('Dept', inplace=True, axis=1)
y_test = test_data.iloc[test_data[test_data.Dept > 100][y_data].dropna().index][y_data]
x_test.shape, y_test.shape
```

```
((30368, 4), (30368, 1))
```

```
x_train.fillna(-999.25, inplace=True)
print()
```

```
x_test['KS'] = np.nan
x_test['BK'] = np.nan
x_test.fillna(-999.25, inplace=True)
print()
```

## Обучение методов

### Random Forest

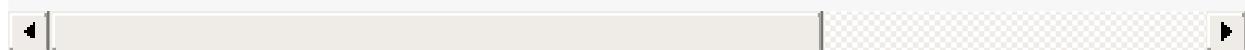
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

```
params = {'n_estimators':10,
          'max_depth':15,
          'class_weight':'balanced',
          'random_state' : seed}
clf = RandomForestClassifier(**params)
clf.fit(x_train, y_train)
clf.score(x_train, y_train), clf.score(x_test, y_test)
```

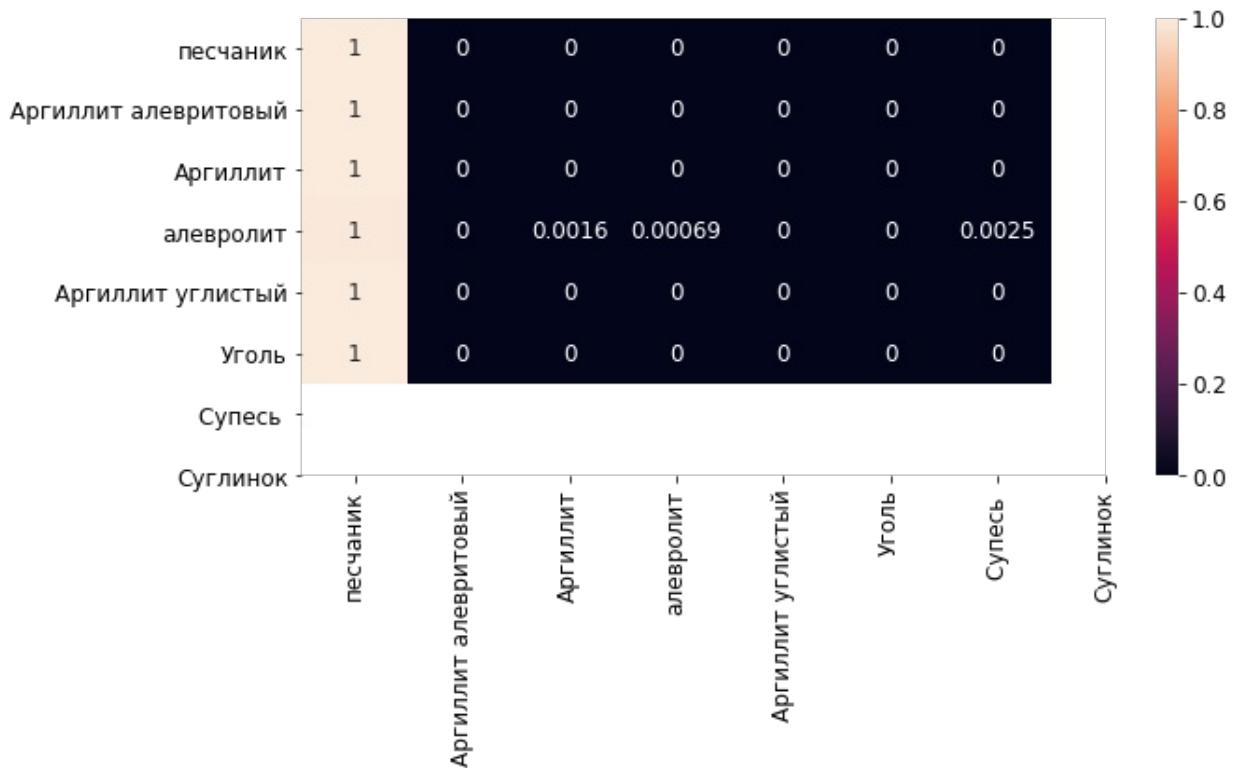
```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python
```

```
(0.79815376879007205, 0.41570073761854581)
```

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]
plt.figure(figsize=(10, 6))
plot_confusion_matrix(confusion_matrix(y_test, clf.predict(x_test)), ticks=ticks, savefile_name='RandomForest_'.join(['{}-{}'.format(k, v) for
```



```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python3.6/m
```



Заполнение "нулями" вообще не работает хд

Поэтому для них надо искать замену либо вообще их удалять

## LogisticRegression

```
from sklearn.preprocessing import StandardScaler
```

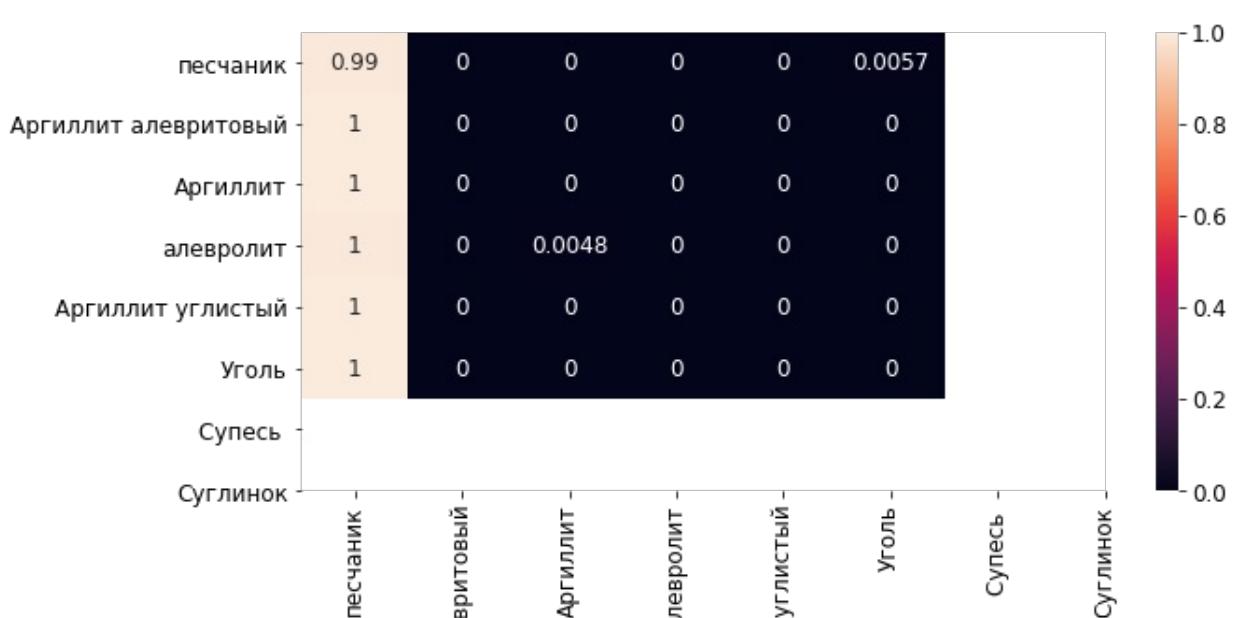
```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
params = {'class_weight': 'balanced',
          'random_state' : seed}
clf = LogisticRegression(**params)
clf.fit(x_train_scaled, y_train)
clf.score(x_train_scaled, y_train), clf.score(x_test_scaled, y_test)
```

```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python  
y = column_or_1d(y, warn=True)
```

```
(0.45150949156803938, 0.4130663856691254)
```

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]
plt.figure(figsize=(10, 6))
plot_confusion_matrix(confusion_matrix(y_test, clf.predict(x_test_scaled)), ticks=ticks,
                      file_name='LogisticRegression_' + '_'.join(['{}-{}'.format(k,
```



no comment

## Заполнение пропущенных значений аппроксимацией

```
seed = 32
```

### Построение аппроксимирующей модели

```
np.random.seed(seed)
deposits_with_all_methods = ['1093', '1462', '1763', '210']
test_reg_deposit = np.random.choice(deposits_with_all_methods)
train_reg_deposits = deposits_with_all_methods
train_reg_deposits.remove(test_reg_deposit)
```

```
train_reg_data = pd.concat([dtfs_first[k] for k in train_reg_deposits], ignore_index=True)
test_reg_data = dtfs_first[test_reg_deposit].dropna()
train_reg_data.shape, test_reg_data.shape
```

```
((30320, 8), (11127, 8))
```

```
train_reg_data = train_reg_data[train_reg_data.Dept > 100]
test_reg_data = test_reg_data[test_reg_data.Dept > 100]
```

```
from itertools import combinations

def find_missing(combos, all_values):
    """Handy function to find target variable based on combinations"""
    for target in all_values:
        if target not in combos:
            return target
```

```
from copy import copy

from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
```

```

to_model = ['GR', 'BK', 'KS', 'GGK(p)']
models_3 = dict()

params = {'n_estimators':10,
          'max_depth':8,
          'random_state' : seed,
          'criterion':'mse'}

for variables in combinations(to_model, 3):
    variables = [v for v in variables]
    target_var = find_missing(variables, to_model)
    x_reg_train = train_reg_data[variables]
    y_reg_train = train_reg_data[target_var]

    scaler = StandardScaler()
    x_reg_train = scaler.fit_transform(x_reg_train)

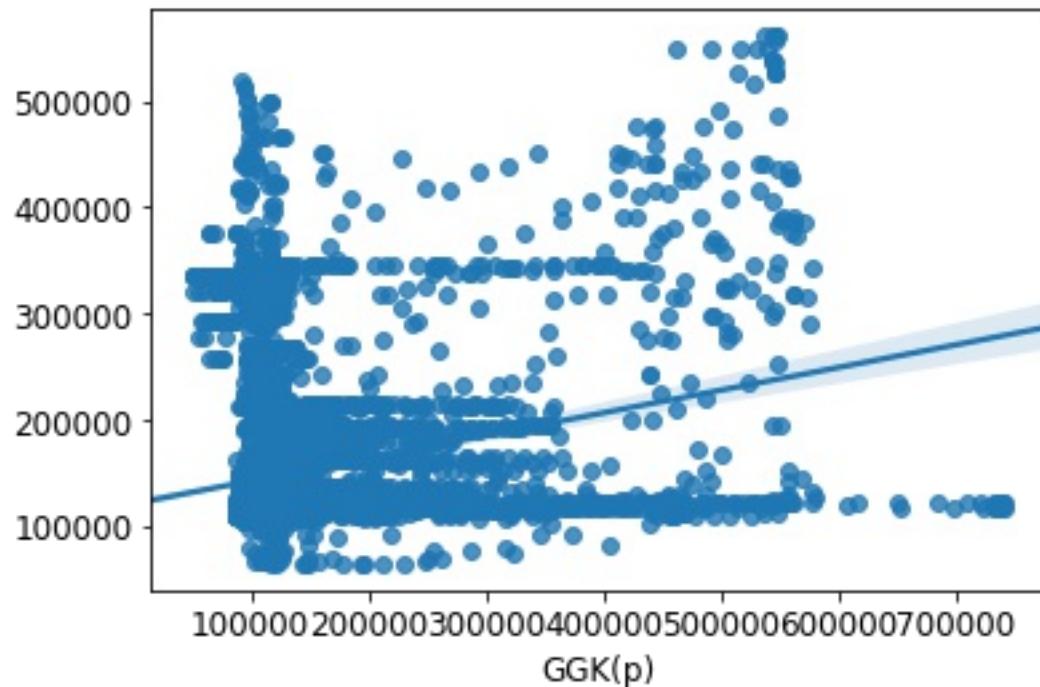
    x_reg_test = test_reg_data[variables]
    y_reg_test = test_reg_data[target_var]

    x_reg_test = scaler.fit_transform(x_reg_test)

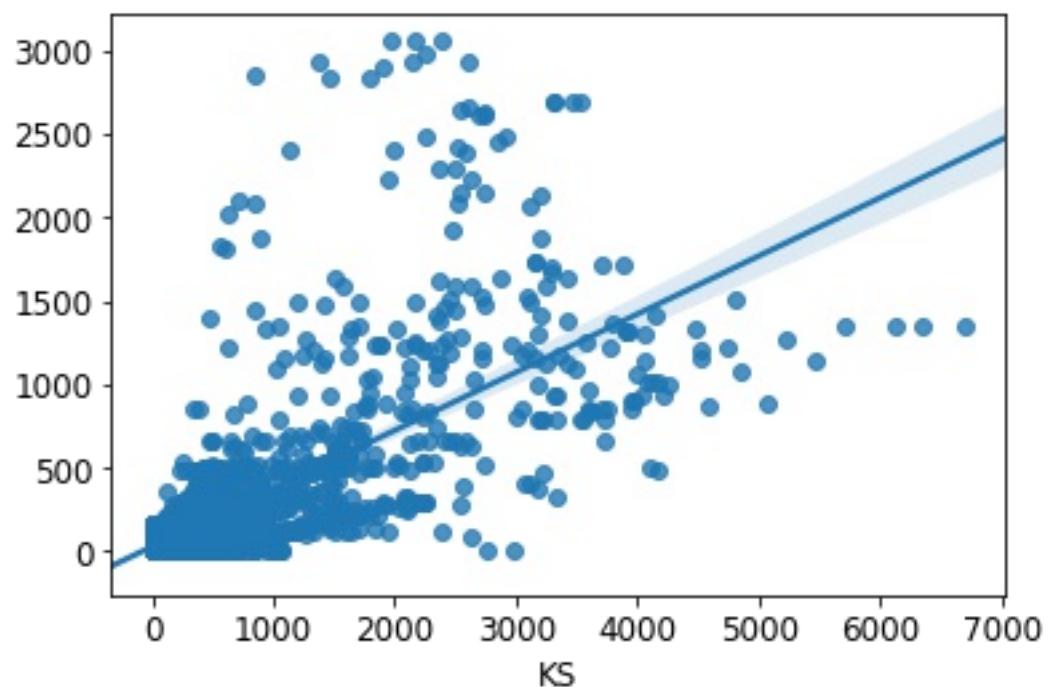
    reg = RandomForestRegressor(**params)
    reg.fit(x_reg_train, y_reg_train)
    print('method: {} train score = {}, test score = {}'.format(target_var,
                                                                  reg.score(x_reg_train),
                                                                  reg.score(x_reg_test)))
    sns.regplot(y_reg_test, reg.predict(x_reg_test))
    plt.show()
    models_3[(target_var, tuple(sorted(variables)))] = (copy(scaler), copy(reg))

```

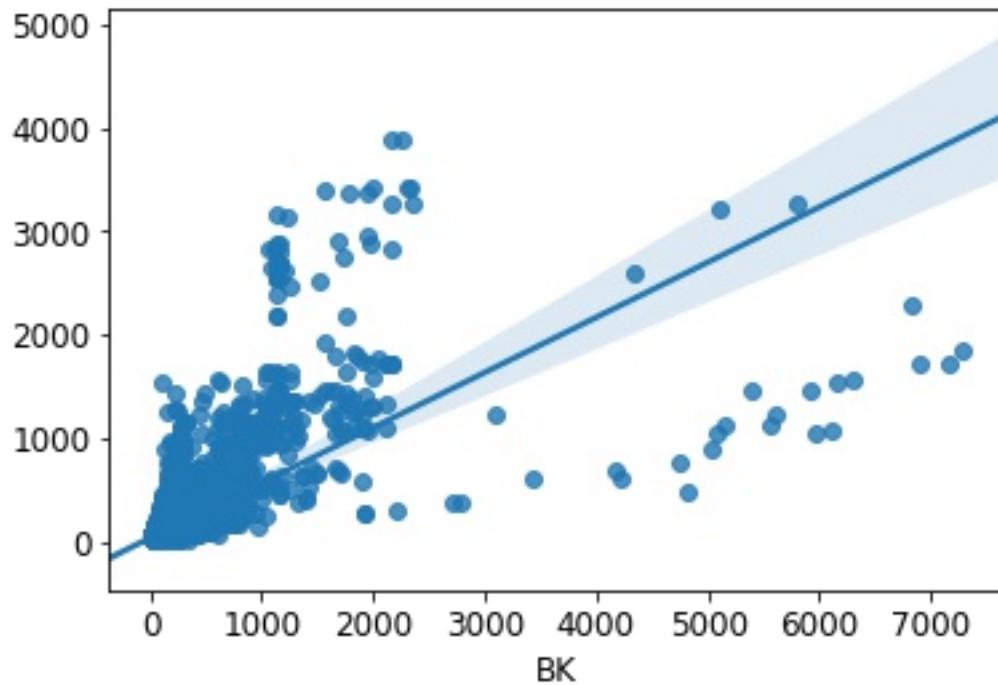
method: GGK(p) train score = 0.5802056483699967, test score = -0.5145779505873616



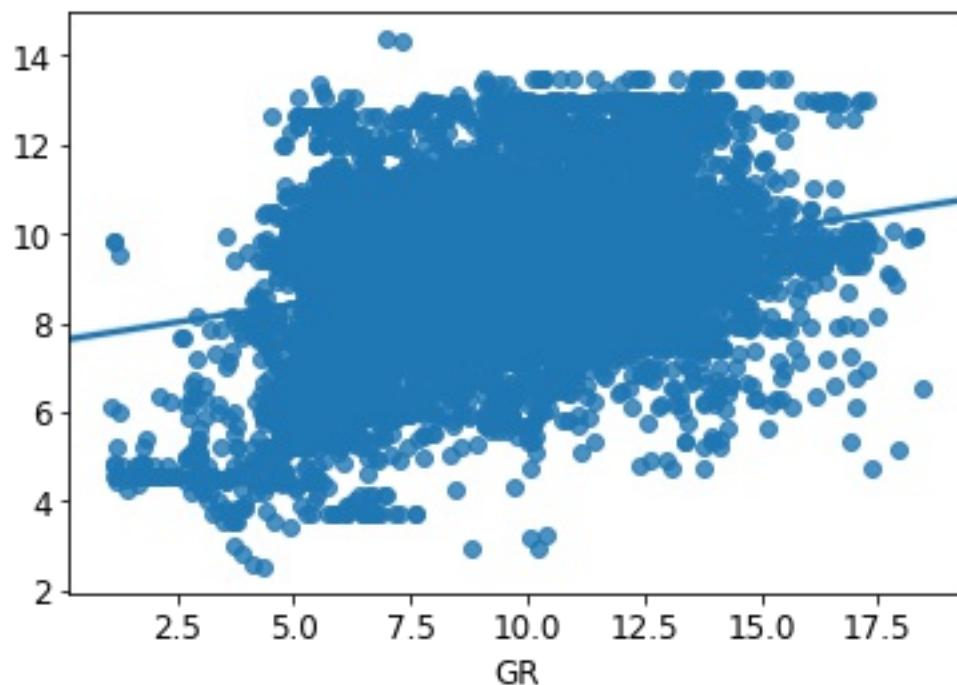
```
method: KS train score = 0.9037521059510103, test score = 0.34813833653230986
```



```
method: BK train score = 0.9032364176778256, test score = 0.3860972609584986
```



```
method: GR train score = 0.6025974603727099, test score = -0.07037775293465542
```



```
to_model = ['GR', 'BK', 'KS', 'GGK(p)']
models_2 = dict()

params = {'n_estimators':10,
          'max_depth':8,
```

```

    'random_state' : seed,
    'criterion':'mse'}

for variables in combinations(to_model, 2):
    for target_var in to_model:
        if target_var not in variables:
            variables = [v for v in variables]
            x_reg_train = train_reg_data[variables]
            y_reg_train = train_reg_data[target_var]

            scaler = StandardScaler()
            x_reg_train = scaler.fit_transform(x_reg_train)

            x_reg_test = test_reg_data[variables]
            y_reg_test = test_reg_data[target_var]

            x_reg_test = scaler.fit_transform(x_reg_test)

            reg = RandomForestRegressor(**params)
            reg.fit(x_reg_train, y_reg_train)
            print('method: {} based on {} train score = {}, test score = {}'.format(
                variables, target_var, reg.score(x_reg_train, y_reg_train), reg.score(x_reg_test, y_reg_test)))

#            sns.regplot(y_reg_test, reg.predict(x_reg_test))
#            plt.show()

models_2[(target_var, tuple(sorted(variables)))] = (copy(scaler), copy(

```

```

method: KS based on ['GR', 'BK'] train score = 0.821730633582307, test score = 0.33
method: GGU(p) based on ['GR', 'BK'] train score = 0.4553978175071438, test score =
method: BK based on ['GR', 'KS'] train score = 0.8719006536386279, test score = -0.
method: GGU(p) based on ['GR', 'KS'] train score = 0.5738986178429137, test score =
method: BK based on ['GR', 'GGU(p)'] train score = 0.729528573910184, test score =
method: KS based on ['GR', 'GGU(p)'] train score = 0.6588285877236999, test score =
method: GR based on ['BK', 'KS'] train score = 0.4938703653940949, test score = -0.
method: GGU(p) based on ['BK', 'KS'] train score = 0.5295153724716111, test score =
method: GR based on ['BK', 'GGU(p)'] train score = 0.5644170774050185, test score =
method: KS based on ['BK', 'GGU(p)'] train score = 0.8791367211021193, test score =
method: GR based on ['KS', 'GGU(p)'] train score = 0.5465178514478811, test score =
method: BK based on ['KS', 'GGU(p)'] train score = 0.8729171079808876, test score =

```

```

to_model = ['GR', 'BK', 'KS', 'GGU(p)']
models_1 = dict()

params = {'n_estimators':10,
          'max_depth':None,
          'random_state' : seed,
          'criterion':'mse'}

```

```

for variables in to_model:
    for target_var in to_model:
        if target_var != variables:
            x_reg_train = train_reg_data[variables].values.reshape(-1, 1)
            y_reg_train = train_reg_data[target_var].values.reshape(-1, 1)

            scaler = StandardScaler()
            x_reg_train = scaler.fit_transform(x_reg_train).reshape(-1, 1)

            x_reg_test = test_reg_data[variables].values.reshape(-1, 1)
            y_reg_test = test_reg_data[target_var].values.reshape(-1, 1)

            x_reg_test = scaler.fit_transform(x_reg_test).reshape(-1, 1)

            reg = RandomForestRegressor(**params)
            reg.fit(x_reg_train, y_reg_train.ravel())
            print('method: {} based on {} train score = {}, test score = {}'.format(
                variables, target_var, reg.score(x_reg_train, y_reg_train), reg.score(x_reg_test, y_reg_test)))
            models_1[(target_var, tuple([variables]))] = (copy(scaler), copy(reg))

```



```

method: BK based on GR train score = 0.6466396313556235, test score = -0.2590664313
method: KS based on GR train score = 0.5472100596204397, test score = 0.02670769195
method: GGK(p) based on GR train score = 0.38937484162239655, test score = -0.13424
method: GR based on BK train score = 0.686791122808751, test score = -0.46985461115
method: KS based on BK train score = 0.9050646475754561, test score = 0.35542109843
method: GGK(p) based on BK train score = 0.7444559220436309, test score = -0.409622
method: GR based on KS train score = 0.6525027021271049, test score = -0.9561370420
method: BK based on KS train score = 0.8714855385172976, test score = 0.08628078150
method: GGK(p) based on KS train score = 0.7906770323965364, test score = -0.934083
method: GR based on GGK(p) train score = 0.7740797505098062, test score = -0.100682
method: BK based on GGK(p) train score = 0.8243323111625926, test score = -0.190956
method: KS based on GGK(p) train score = 0.816290184258504, test score = -0.1583329

```



По Графиками, а также коэффициентам детерминации видно, что можно моделировать интересующие нас, пропущенные, методы каротажа: BK и KS . GGK(p) и GK Не столь хорошо моделируются. я Попробовал применить  $\log(y+1)$ , к целевой переменной, это не помогло.

Моделировать один метод через другой выглядит не очень здорово, но посмотрим на результаты классификации при добавлении этих значений.

## Train/test split

```
seed = 30
```

```
X_data = ['GR', 'BK', 'KS', 'GGK(p)', 'DS', 'REZ', 'Dept']
X_test_data = ['GR', 'GGK(p)', 'DS', 'REZ', 'Dept']
y_data = ['LIT']
```

```
excluded_dtsfs = ['875']
np.random.seed(seed)
test_dtf_number = np.random.choice([k for k in dtsfs_first.keys() if k not in excluded_dtsfs])
excluded_dtsfs.append(test_dtf_number)
print(excluded_dtsfs[1])
```

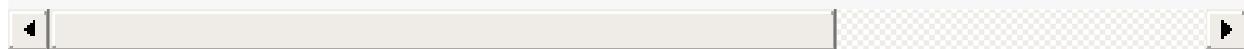
```
1742
```

```
train_data = pd.concat([d for k, d in dtsfs_first.items() if k not in excluded_dtsfs]
test_data = pd.concat([d for k, d in dtsfs_second.items() if k not in excluded_dtsfs]
train_data[X_data].shape, test_data[X_test_data].shape
```

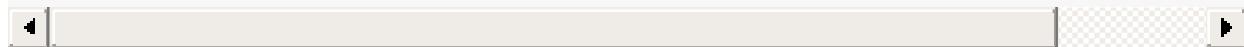
```
((331632, 7), (65457, 5))
```

```
def fill_vals(row):
    r = row
    known_methods = []
    unknown_methods = []
    for k in to_model:
        if k in r.keys():
            if np.isnan(r[k]):
                unknown_methods.append(k)
            else:
                known_methods.append(k)
        else:
            unknown_methods.append(k)
    if len(known_methods) == 1:
```

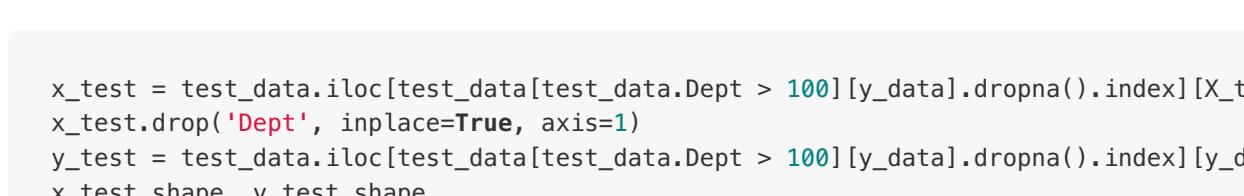
```
for um in unknown_methods:
    x = models_1[(um, tuple(sorted(known_methods)))] [0].transform(row[known])
    pred = models_1[(um, tuple(sorted(known_methods)))] [1].predict(x)
    r[um] = pred[0]
elif len(known_methods) == 2:
    for um in unknown_methods:
        x = models_2[(um, tuple(sorted(known_methods)))] [0].transform(row[known])
        pred = models_2[(um, tuple(sorted(known_methods)))] [1].predict(x.reshape(1,-1))
        r[um] = pred[0]
elif len(known_methods) == 3:
    for um in unknown_methods:
        x = models_3[(um, tuple(sorted(known_methods)))] [0].transform(row[known])
        pred = models_3[(um, tuple(sorted(known_methods)))] [1].predict(x.reshape(1,-1))
        r[um] = pred[0]
return r
```



```
# Берем данные глубже 100 метров, потому что см. анализ данных
x_train = train_data.iloc[train_data[train_data.Dept > 100][y_data].dropna().index]
x_train.drop('Dept', inplace=True, axis=1)
y_train = train_data.iloc[train_data[train_data.Dept > 100][y_data].dropna().index]
x_train.shape, y_train.shape
```



```
((222724, 6), (222724, 1))
```



```
((30368, 4), (30368, 1))
```



```
if 'KS' not in x_test.columns:
    x_test['KS'] = np.nan
```

```
if 'BK' not in x_test.columns:  
    x_test['BK'] = np.nan  
x_test = x_test.apply(fill_vals, axis=1)  
print()
```

## Обучение методов

### Random Forest

```
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear_model import LogisticRegression  
from sklearn.metrics import confusion_matrix
```

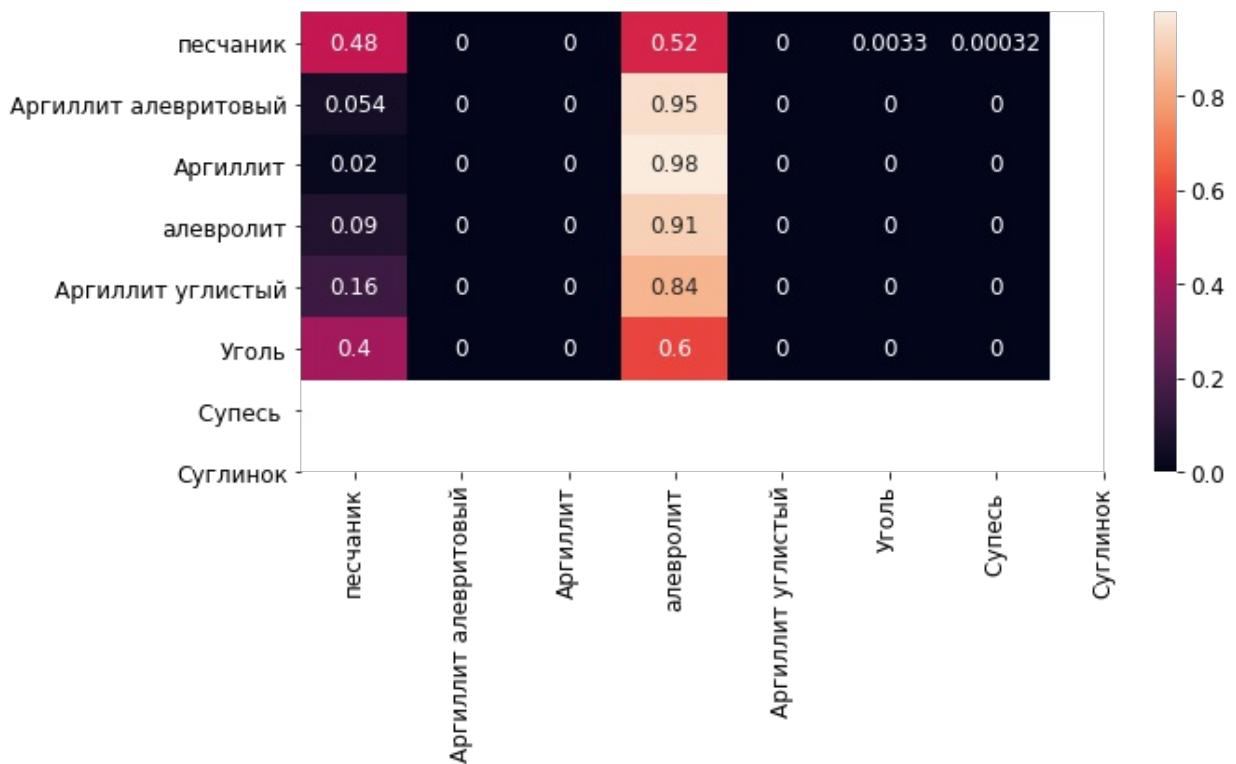
```
params = {'n_estimators':10,  
          'max_depth':None,  
          'class_weight':'balanced',  
          'random_state' : seed}  
clf = RandomForestClassifier(**params)  
clf.fit(x_train.dropna(), y_train.loc[x_train.dropna().index])  
print(clf.score(x_train.dropna(), y_train.loc[x_train.dropna().index]),  
      clf.score(x_test.dropna(), y_test.loc[x_test.dropna().index]))
```

/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python

0.995618641779 0.545493533914

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]  
plt.figure(figsize=(10, 6))  
plot_conusion_matrix(confusion_matrix(y_test.loc[x_test.dropna().index], clf.predict(x_test.dropna())),  
                     file_name='RandomForest_ ' + '_'.join(['{}-{}'.format(k, v) for
```

/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python



Заполнение "нулями" вообще не работает хд

Поэтому для них надо искать замену либо вообще их удалять

## LogisticRegression

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train.dropna())
x_test_scaled = scaler.transform(x_test.dropna())
```

```
params = {'class_weight':'balanced',
          'random_state' : seed}
clf = LogisticRegression(**params)
clf.fit(x_train_scaled, y_train.loc[x_train.dropna().index])
clf.score(x_train_scaled, y_train.loc[x_train.dropna().index]), clf.score(x_test_sc
```

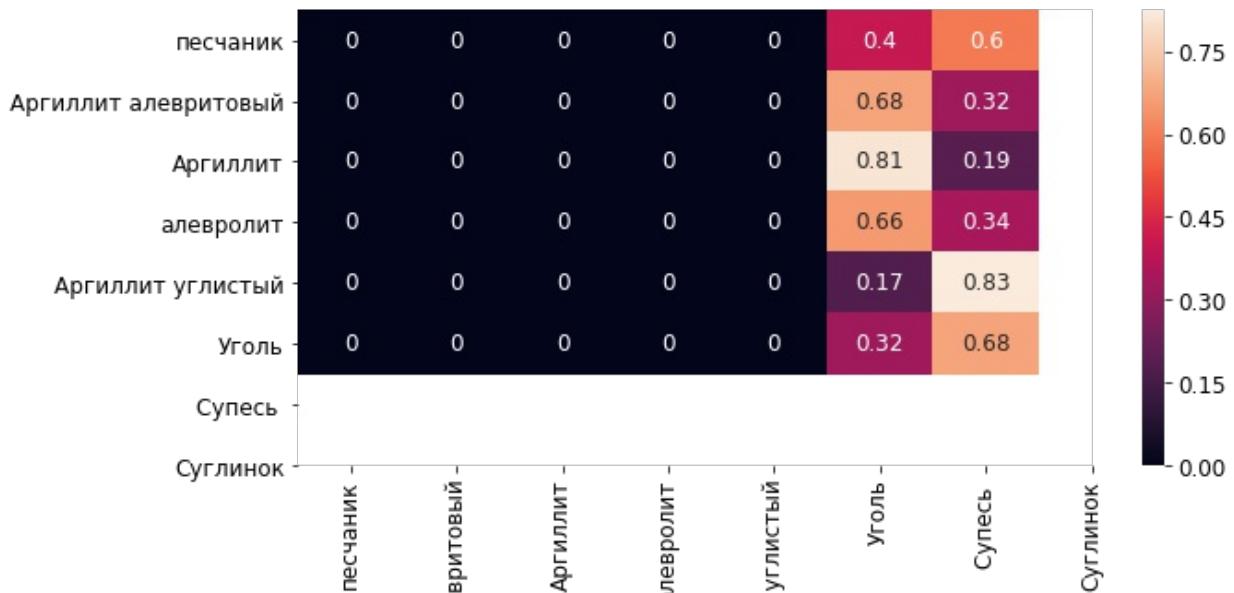
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python3.8/site-packages/sklearn/preprocessing/\_base.py:20: DeprecationWarning: The 'n\_iter' parameter is deprecated. Use 'max\_iter' instead.

```
y = column_or_1d(y, warn=True)
```

```
(0.54366353569606574, 0.0013855898653998416)
```

```
ticks = [lithology[num] for num in [1, 2, 3, 5, 6, 7, 8, 9]]  
plt.figure(figsize=(10, 6))  
plot_conusion_matrix(confusion_matrix(y_test.loc[x_test.dropna().index], clf.predict(x_test)),  
file_name='LogisticRegression_' + '_'.join(['{}-{}'.format(k,
```

```
/Volumes/Media/Documents/Programming/Git/(DD)-gitlab-projects/carrot/venv/lib/python
```



## Выводы:

К сожалению, магия не случилась и точность не повысилась.

Но тут может быть другая проблема быть причиной:

- При моделировании методов, коэффициент детерминации во многих случаях очень маленький, поэтому имеет смысл обратить на это внимание.
  - Пограть с масштабом данных;
  - Попробовать иные методы нелинейной регрессии.
- Мы также добавляем еще больше лишних значений для и так большого количества литологий, мб в этом проблема.

## Изучение важности KS

```
from itertools import chain
```

```
dtfs_with_ks = {name:dft for name, dft in chain(dtfs_first.items()) if 'KS' in dft.
```

```
for n, dft in dtfs_with_ks.items():
    check_values_for_str(dft, 'LIT', n)
```

```
test_dtf_number = np.random.choice([k for k in dtfs_with_ks.keys()])
test_dtf_number
```

```
'1462'
```

```
dft_with_ks = pd.concat([dft for n, dft in dtfs_with_ks.items() if n != test_dtf_nu
```

```
dft_with_ks.shape, dft_with_ks.dropna().shape
```

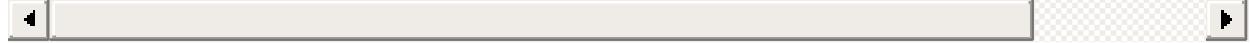
```
((49069, 2), (27652, 2))
```

```
X = dtf_with_ks.dropna()['KS'].values.reshape((-1, 1))
Y = dtf_with_ks.dropna()['LIT'].values.reshape((-1, 1))
```

```
X.shape, Y.shape
```

```
((27652, 1), (27652, 1))
```

```
x_test = dtfs_with_ks[test_dtf_number][['KS', 'LIT']].dropna()['KS'].values.reshape(-1, 1)
y_test = dtfs_with_ks[test_dtf_number][['KS', 'LIT']].dropna()['LIT'].values.reshape(-1, 1)
```



## Классификация

```
from copy import copy
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import scale, minmax_scale, maxabs_scale, robust_scale
```



```
def test_scaler_and_clf(dtfs_with_ks, y_train, x_test, y_test, clf_dict, scaler_dict):
    dtf_scores_train = pd.DataFrame(columns=clf_dict.keys(), index=scaler_dict.keys())
    dtf_scores_test = pd.DataFrame(columns=clf_dict.keys(), index=scaler_dict.keys())
    for scaler_name, scaler_func in scaler_dict.items():
        x_train_scaled = np.vstack(map(scaler_func, [dtf[['KS', 'LIT']].dropna()]))
        x_test_scaled = scaler_func(x_test)
        for clf_name, clf_init in clf_dict.items():
            clf = copy(clf_init)
            clf.fit(x_train_scaled, y_train)
            if verbose:
                print('clf: {} \n scaler: {} \n score_train: {} \n score_test: {}'.format(
                    clf,
                    scaler_name,
                    clf.score(x_train_scaled, y_train),
                    clf.score(x_test_scaled, y_test)))
            dtf_scores_train.loc[scaler_name][clf_name] = clf.score(x_train_scaled, y_train)
            dtf_scores_test.loc[scaler_name][clf_name] = clf.score(x_test_scaled, y_test)
    return dtf_scores_train, dtf_scores_test
```



```

scalers = {'No scale' : lambda x: x,
           'StandartScaler':scale,
           'MinMax scale' : minmax_scale,
           'MaxAbs scale' : maxabs_scale,
           'Robust scale' : robust_scale}

clfs = {'RandomForestClassifier' : RandomForestClassifier(class_weight='balanced'),
        'GradientBoostingClassifier' : GradientBoostingClassifier(),
        'LogisticRegression' : LogisticRegression(C=3, class_weight='balanced')}

```

```
dtf_score_train, dtf_score_test = test_scaler_and_clf(dtfs_with_ks, Y, x_test, y_te
```

```
# dtf_score_train.to_excel('scores_train.xlsx')
dtf_score_train
```

	<b>RandomForestClassifier</b>	<b>GradientBoostingClassifier</b>	<b>Log</b>
<b>No scale</b>	0.735643	0.611529	0.26
<b>StandartScaler</b>	0.750759	0.621763	0.54
<b>MinMax scale</b>	0.728808	0.581766	0.31
<b>MaxAbs scale</b>	0.735318	0.59323	0.31
<b>Robust scale</b>	0.739621	0.613988	0.42

```
# dtf_score_test.to_excel('scores_test.xlsx')
dtf_score_test
```

	<b>RandomForestClassifier</b>	<b>GradientBoostingClassifier</b>	<b>Log</b>
<b>No scale</b>	0.407024	0.525439	0.27
<b>StandartScaler</b>	0.382367	0.447378	0.41

<b>MinMax scale</b>	0.202069	0.268403	0.49
<b>MaxAbs scale</b>	0.400289	0.527123	0.19
<b>Robust scale</b>	0.384111	0.471795	0.26