Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №1
по дисциплине
«Методы машинного обучения»
на тему
«Разведочный анализ данных. Исследование и визуализация данных»

Выполнил:
студент группы ИУ5-23М
Умряев Д. Т.

_____

Москва — 2020 г.

1. Цель лабораторной работы

Изучить различные методы визуализация данных [1].

2. Задание

Требуется выполнить следующие действия [1]:

- Выбрать набор данных (датасет).
- Создать ноутбук, который содержит следующие разделы:
    1. Текстовое описание выбранного Вами набора данных.
    2. Основные характеристики датасета.
    3. Визуальное исследование датасета. Необходимо использовать не менее 2 различных библиотек и не менее 5 графиков.
    4. Информация о корреляции признаков.
- Сформировать отчет и разместить его в своем репозитории на github.

3. Ход выполнения работы

3.1. Текстовое описание набора данных

В качестве набора данных мы будем использовать набор данных температуры электрического мотора - https://www.kaggle.com/wkirgsn/electric-motor-temperature [2]

Набор данных содержит несколько данных датчиков, собранных с синхронного двигателя с постоянными магнитами (PMSM), установленного на испытательном стенде. PMSM представляет модель-прототип немецкого производителя. Измерения на стенде были собраны отделом LEA в университете Падерборн. Этот набор данных является слегка анонимным.

Все записи сэмплированы с частотой 2 Гц. Набор данных состоит из нескольких сеансов измерений, которые можно отличить друг от друга по столбцу «profile_id». Сеанс измерения может длиться от одного до шести часов.

Двигатель возбуждается ручными рабочими циклами, обозначающими опорную скорость двигателя и опорный крутящий момент. Токи в d/q-координатах (столбцы «id» и «iq») и напряжения в d/q-координатах (столбцы «ud» и «uq») являются результатом стандартной стратегии управления, которая пытается следовать эталонной скорости и крутящему моменту. Столбцы «motor_speed» и «torque» являются результирующими величинами, достигнутыми этой стратегией, полученными из заданных токов и напряжений.

Большинство циклов вождения обозначают случайные движения в плоскости скорость-крутящий момент, чтобы имитировать реальные циклы вождения в более точной степени, чем постоянные возбуждения, подъемы и спады.

Датасет состоит из одного файла:

- pmsm_temperature_data.csv

Файл содержит следующие колонки:

- ambient - температура окружающей среды, измеренная термодатчиком, расположенным близко к статору;
- coolant - температура охлаждающей жидкости. Мотор с водяным охлаждением. Измерение производится при оттоке;

- u_d - напряжение d-компонента;
- u_q - q-компонент напряжения;
- motor_speed - скорость двигателя;
- torque - крутящий момент, вызванный током;
- i_d - сила тока d-компонента;
- i_q - сила тока q-компонента;
- pm - температура поверхности постоянного магнита, представляющая температуру ротора. Это было измерено с помощью инфракрасной термографии;
- stator_yoke - температура ярма статора, измеренная термодатчиком;
- stator_tooth - температура зубца статора, измеренная термодатчиком;
- stator_winding - температура обмотки статора, измеренная термодатчиком;
- profile_id - Каждый сеанс измерения имеет уникальный идентификатор. Убедитесь, что не пытаетесь оценить от одного сеанса к другому, поскольку они сильно независимы.

3.1.1. Импорт библиотек

Импортируем библиотеки с помощью команды import.

[2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
```

Настроим отображение графиков [3, 4]:

[3]:
```
# Enable inline plots
%matplotlib inline

# Set plot style
sns.set(style="ticks")

# Set plots formats to save high resolution PNG
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

Зададим ширину текстового представления данных, чтобы в дальнейшем текст в отчёте влезал на A4 [5]:

[4]:
```
pd.set_option("display.width", 70)
```

3.1.2. Загрузка данных

Загрузим файлы датасета в помощью библиотеки Pandas.

Не смотря на то, что файлы имеют расширение txt они представляют собой данные в формате CSV (https://ru.wikipedia.org/wiki/CSV). Часто в файлах такого формата в качестве разделителей используются символы “,”, “;” или табуляция. Поэтому вызывая метод read_csv всегда стоит явно указывать разделитель данных с помощью параметра sep. Чтобы узнать какой разделитель используется в файле его рекомендуется предварительно посмотреть в любом текстовом редакторе.

[5]:
```
# Будем анализировать данные только на обучающей выборке
data = pd.read_csv('pmsm_temperature_data.csv', sep=",", nrows=1000,
                   usecols=['ambient','coolant','u_d','u_q',
                            'motor_speed','torque','i_d','i_q',
                            'pm','stator_yoke','stator_tooth',
                            'stator_winding'])
```

3.2. Основные характеристики датасета

[6]:
```
# Первые 5 строк датасета
data.head()
```

[6]:

	ambient	coolant	u_d	u_q	motor_speed	torque	i_d	i_q	pm	stator_yoke	stator_tooth	stator_winding
0	-0.752143	-1.118446	0.327935	-1.297858	-1.222428	-0.250182	1.029572	-0.245860	-2.522071	-1.831422	-2.066143	-2.018033
1	-0.771263	-1.117021	0.329665	-1.297686	-1.222429	-0.249133	1.029509	-0.245832	-2.522418	-1.830969	-2.064859	-2.017631
2	-0.782892	-1.116681	0.332771	-1.301822	-1.222428	-0.249431	1.029448	-0.245818	-2.522673	-1.830400	-2.064073	-2.017343
3	-0.780935	-1.116764	0.333700	-1.301852	-1.222430	-0.248636	1.032845	-0.246955	-2.521639	-1.830333	-2.063137	-2.017632
4	-0.774043	-1.116775	0.335206	-1.303118	-1.222429	-0.248701	1.031807	-0.246610	-2.521900	-1.830498	-2.062795	-2.018145

[7]:
```
# Размер датасета - 998070 строк, 12 колонок
data.shape
```

[7]:
```
(1000, 12)
```

[8]:
```
total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 1000
```

[9]:
```
# Список колонок
data.columns
```

[9]:
```
Index(['ambient', 'coolant', 'u_d', 'u_q', 'motor_speed', 'torque',
       'i_d', 'i_q', 'pm', 'stator_yoke', 'stator_tooth',
       'stator_winding'],
```

```
      dtype='object')
```

[10]:
```
# Список колонок с типами данных
data.dtypes
```

[10]:
```
ambient           float64
coolant           float64
u_d               float64
u_q               float64
motor_speed       float64
torque            float64
i_d               float64
i_q               float64
pm                float64
stator_yoke       float64
stator_tooth      float64
stator_winding    float64
dtype: object
```

[11]:
```
# Проверим наличие пустых значений
# Цикл по колонкам датасета
for col in data.columns:
    # Количество пустых значений - все значения заполнены
    temp_null_count = data[data[col].isnull()].shape[0]
    print('{} - {}'.format(col, temp_null_count))
```

```
ambient - 0
coolant - 0
u_d - 0
u_q - 0
motor_speed - 0
torque - 0
i_d - 0
i_q - 0
pm - 0
stator_yoke - 0
stator_tooth - 0
stator_winding - 0
```

[12]:
```
# Основные статистические характеристки набора данных
data.describe()
```

[12]:

	ambient	coolant	u_d	u_q	\
count	1000.000000	1000.000000	1000.000000	1000.000000	
mean	-0.877601	-1.068030	0.190565	1.611280	
std	0.098472	0.018251	0.025801	0.430905	
min	-1.013299	-1.118446	0.178976	-1.306618	
25%	-0.948757	-1.080291	0.182582	1.674642	
50%	-0.909092	-1.065906	0.184389	1.677505	
75%	-0.819927	-1.053867	0.186992	1.680015	

```
max      -0.592720    -1.039449     0.336557     1.762809

       motor_speed       torque          i_d          i_q  \
count  1000.000000  1000.000000  1000.000000  1000.000000
mean      1.901647    -0.267095    -0.740862    -0.255367
std       0.531425     0.013907     0.336035     0.002491
min      -1.222434    -0.404643    -0.878109    -0.257432
25%       2.024116    -0.266207    -0.835955    -0.256361
50%       2.024118    -0.265140    -0.813416    -0.255938
75%       2.024119    -0.264251    -0.796285    -0.255520
max       2.024125    -0.247513     1.032845    -0.245577

                pm  stator_yoke  stator_tooth  stator_winding
count  1000.000000  1000.000000   1000.000000     1000.000000
mean     -1.612218    -1.310481     -1.184094       -1.011330
std       0.509711     0.293293      0.427544        0.478505
min      -2.524219    -1.834688     -2.066143       -2.019973
25%      -2.034048    -1.560341     -1.461220       -1.322479
50%      -1.533538    -1.223762     -1.067903       -0.844669
75%      -1.158214    -1.046125     -0.797070       -0.601496
max      -0.882228    -0.951761     -0.761951       -0.486731
```

3.3. Визуальное исследование датасета

Для визуального исследования могут быть использованы различные виды диаграмм, мы построим только некоторые варианты диаграмм, которые используются достаточно часто.

3.3.1. Диаграмма рассеяния

Позволяет построить распределение двух колонок данных и визуально обнаружить наличие зависимости. Не предполагается, что значения упорядочены (например, по времени).

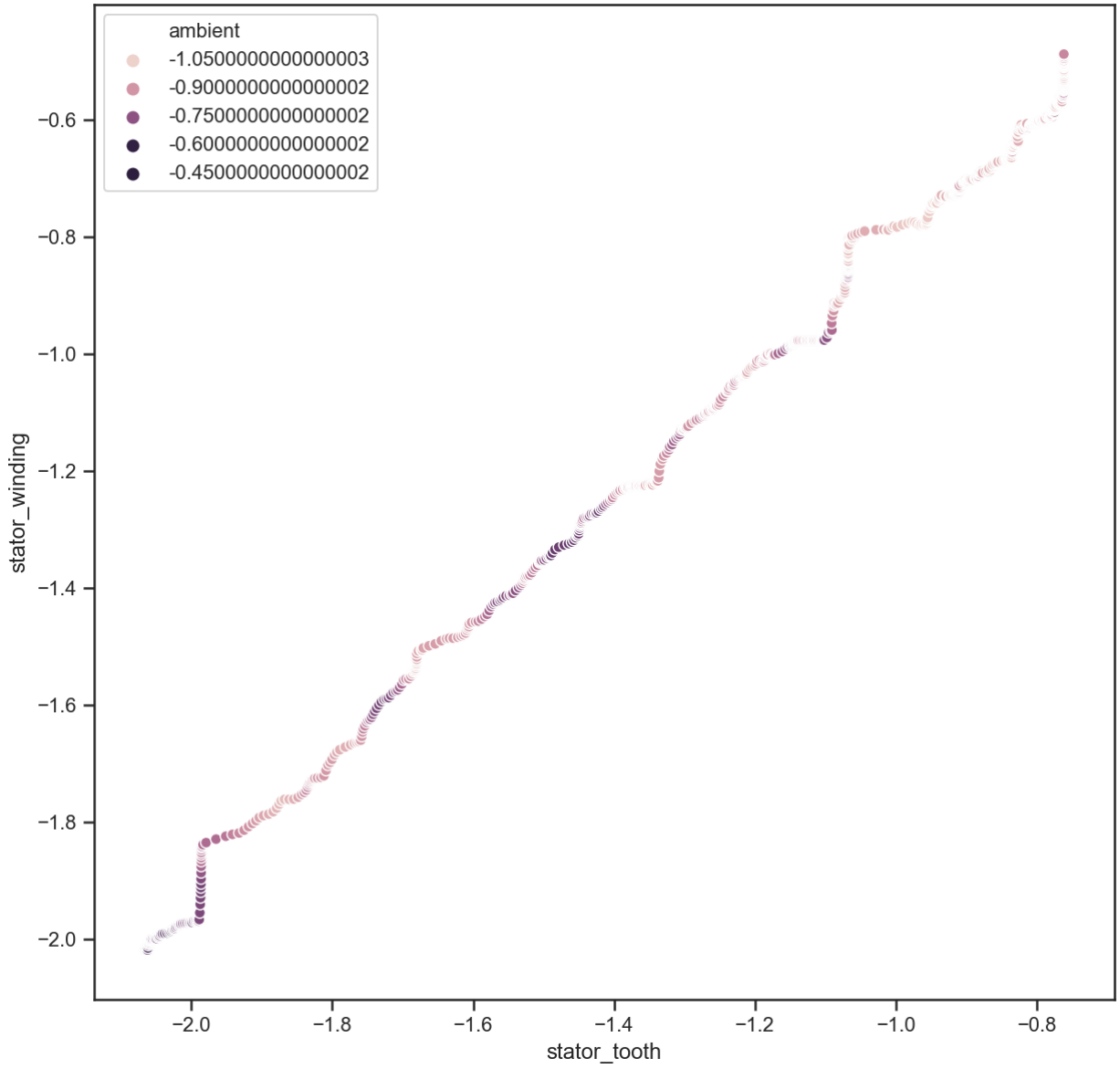
[13]:
```
px.scatter(data, x="stator_tooth", y="stator_winding", trendline="ols")
```

Можно видеть что между полями stator_tooth и stator_winding пристутствует почти линейная зависимость.

Посмотрим насколько на эту зависимость влияет целевой признак.

[14]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(ax=ax, x='stator_tooth', y='stator_winding', data=data,
                hue='ambient')
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x262e0ef5dc8>
```

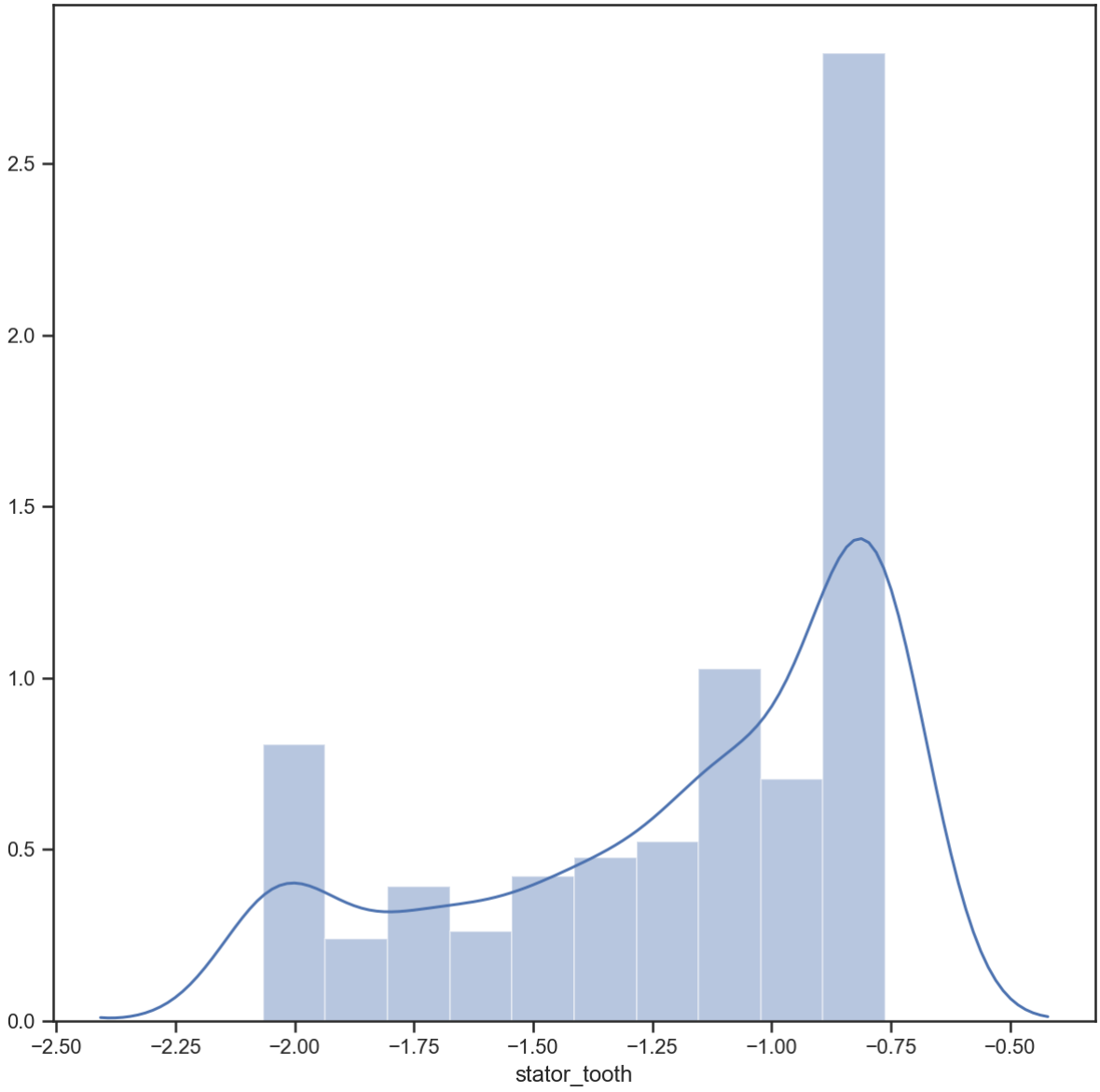


3.3.2. Гистограмма

Позволяет оценить плотность вероятности распределения данных.

[15]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.distplot(data['stator_tooth'])
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x262e129dc08>
```



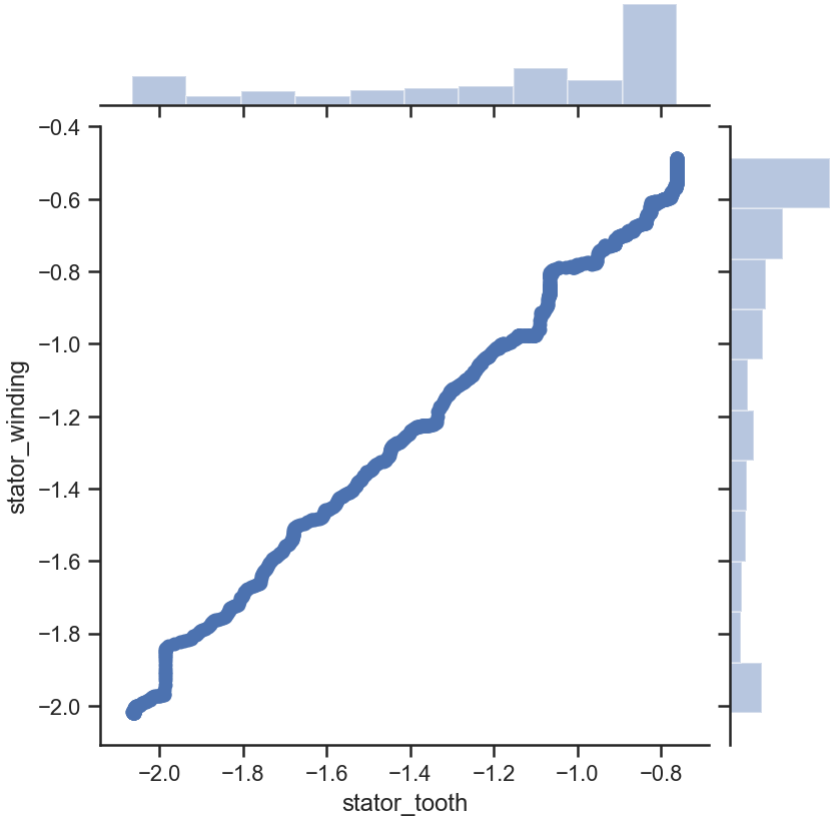
[16]:
```
go.Figure(go.Histogram2dContour(x=data['stator_tooth'],
                                y=data['stator_winding'],
                                colorscale = 'Blues'))
```

3.3.3. Jointplot

Комбинация гистограмм и диаграмм рассеивания.

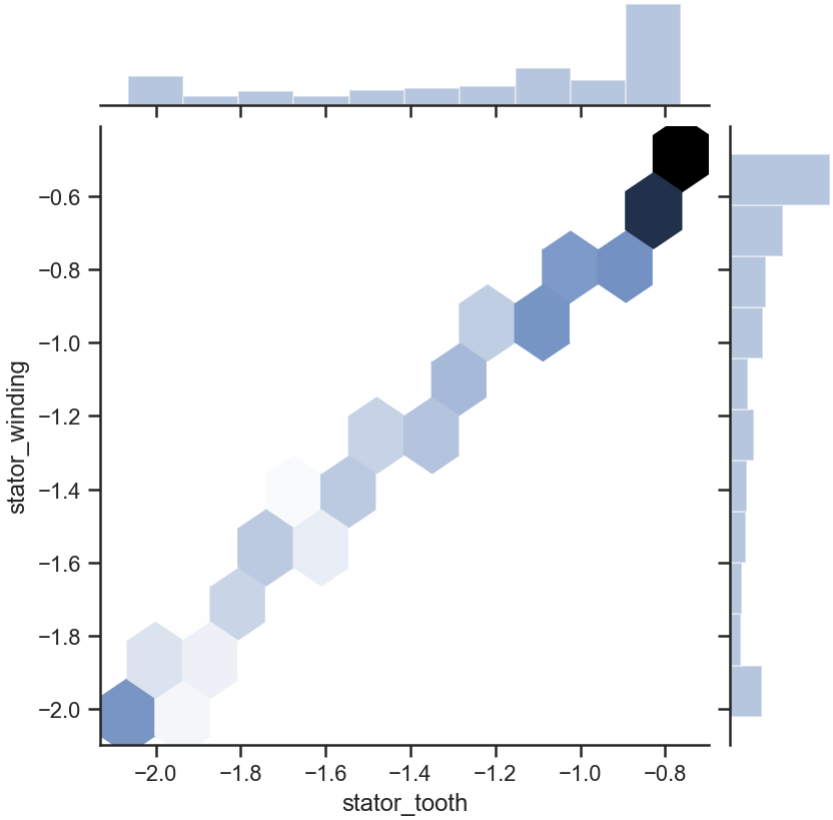
[17]:
```
sns.jointplot(x='stator_tooth', y='stator_winding', data=data)
```

[17]:
```
<seaborn.axisgrid.JointGrid at 0x262e10fec08>
```



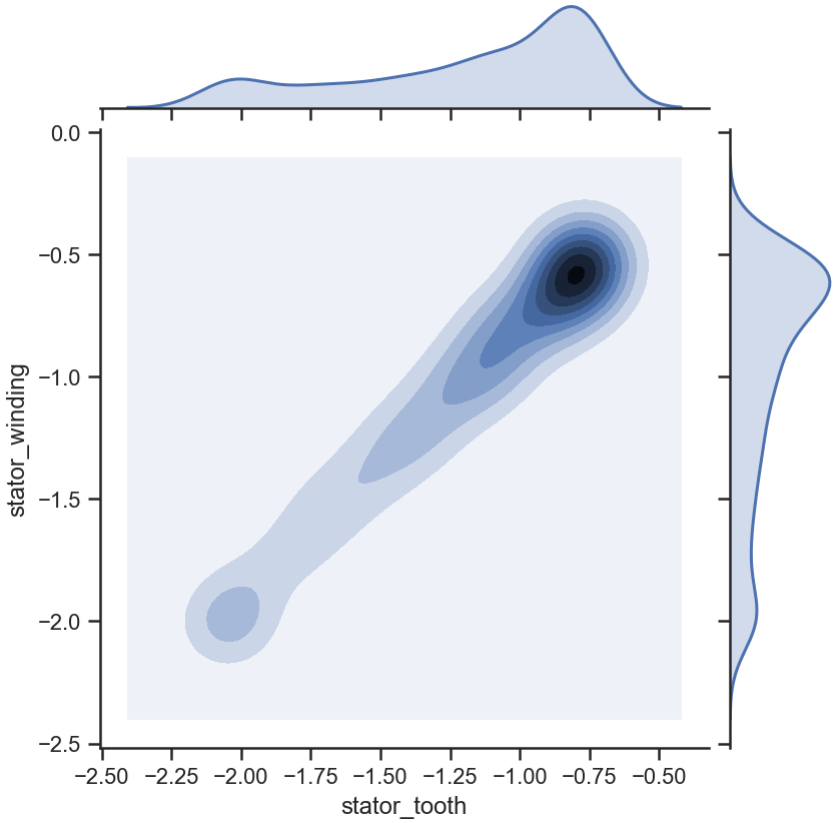
[18]:
```
sns.jointplot(x='stator_tooth', y='stator_winding', data=data,
              kind="hex")
```

[18]: <seaborn.axisgrid.JointGrid at 0x262e308de88>



[19]:
```
sns.jointplot(x='stator_tooth', y='stator_winding', data=data,
              kind="kde")
```

[19]: <seaborn.axisgrid.JointGrid at 0x262e08fd888>



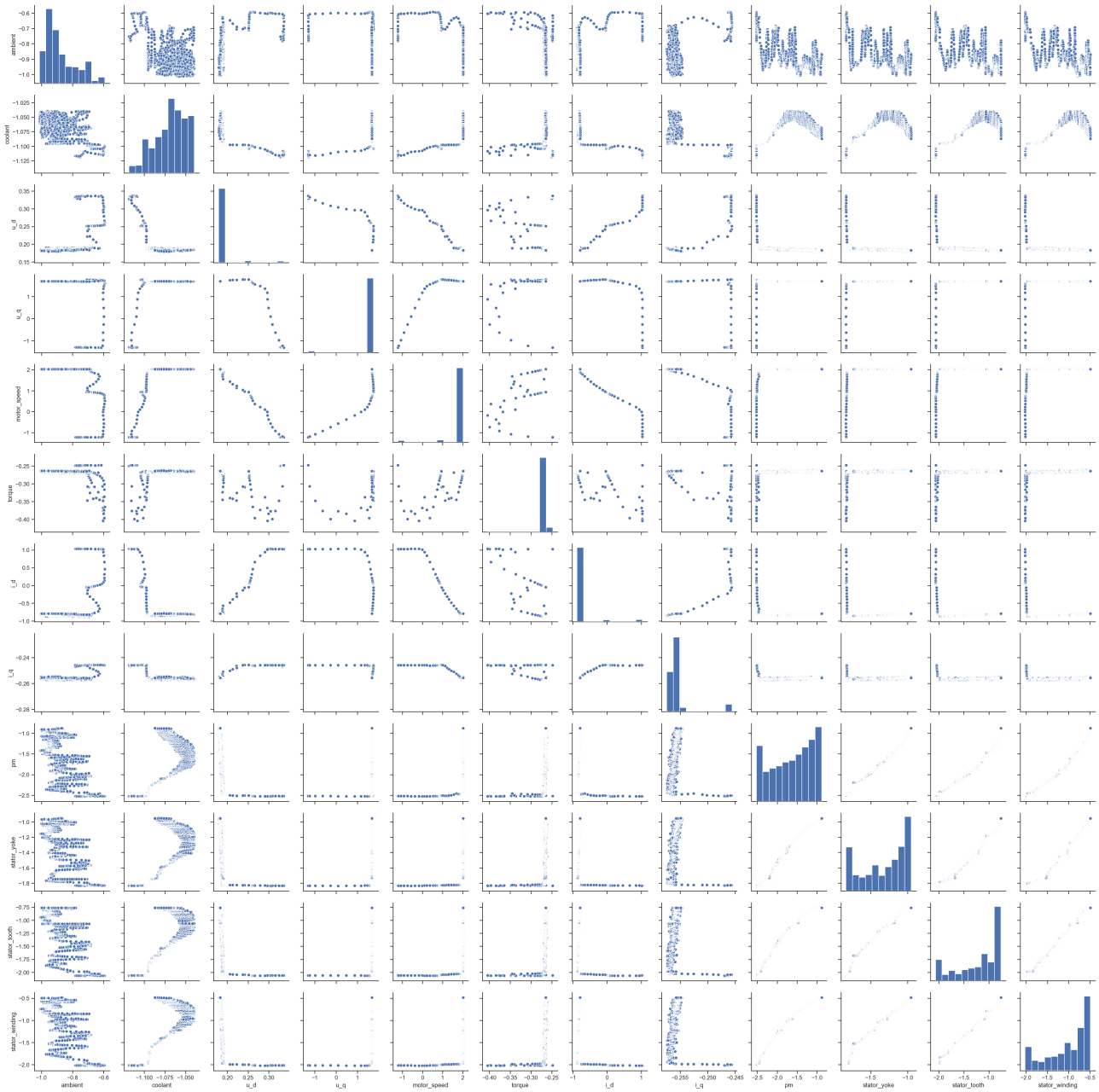
3.3.4. “Парные диаграммы”

Комбинация гистограмм и диаграмм рассеивания для всего набора данных.

Выводится матрица графиков. На пересечении строки и столбца, которые соответствуют двум показателям, строится диаграмма рассеивания. В главной диагонали матрицы строятся гистограммы распределения соответствующих показателей.

[20]:
```
sns.pairplot(data)
```

[20]:
```
<seaborn.axisgrid.PairGrid at 0x262e0a5d608>
```



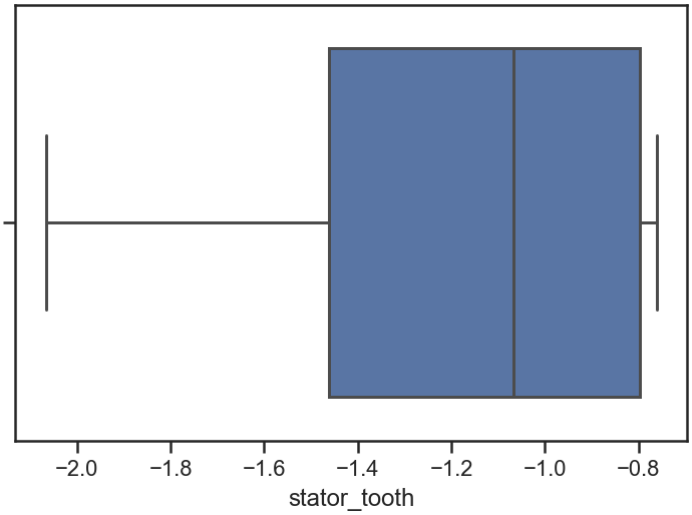
С помощью параметра “hue” возможна группировка по значениям какого-либо признака.

3.3.5. Ящик с усами

Отображает одномерное распределение вероятности.

[21]:
```
sns.boxplot(x=data['stator_tooth'])
```

[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x262e98e4e48>
```



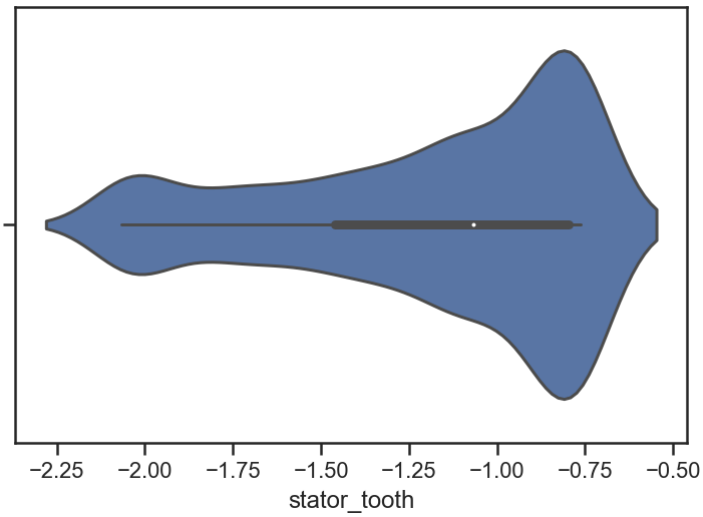
[22]:
```
# По вертикали
px.box(data, y='stator_tooth')
```

3.3.6. Violin plot

Похоже на предыдущую диаграмму, но по краям отображаются распределения плотности - https://en.wikipedia.org/wiki/Kernel_density_estimation

[23]:
```
sns.violinplot(x=data['stator_tooth'])
```

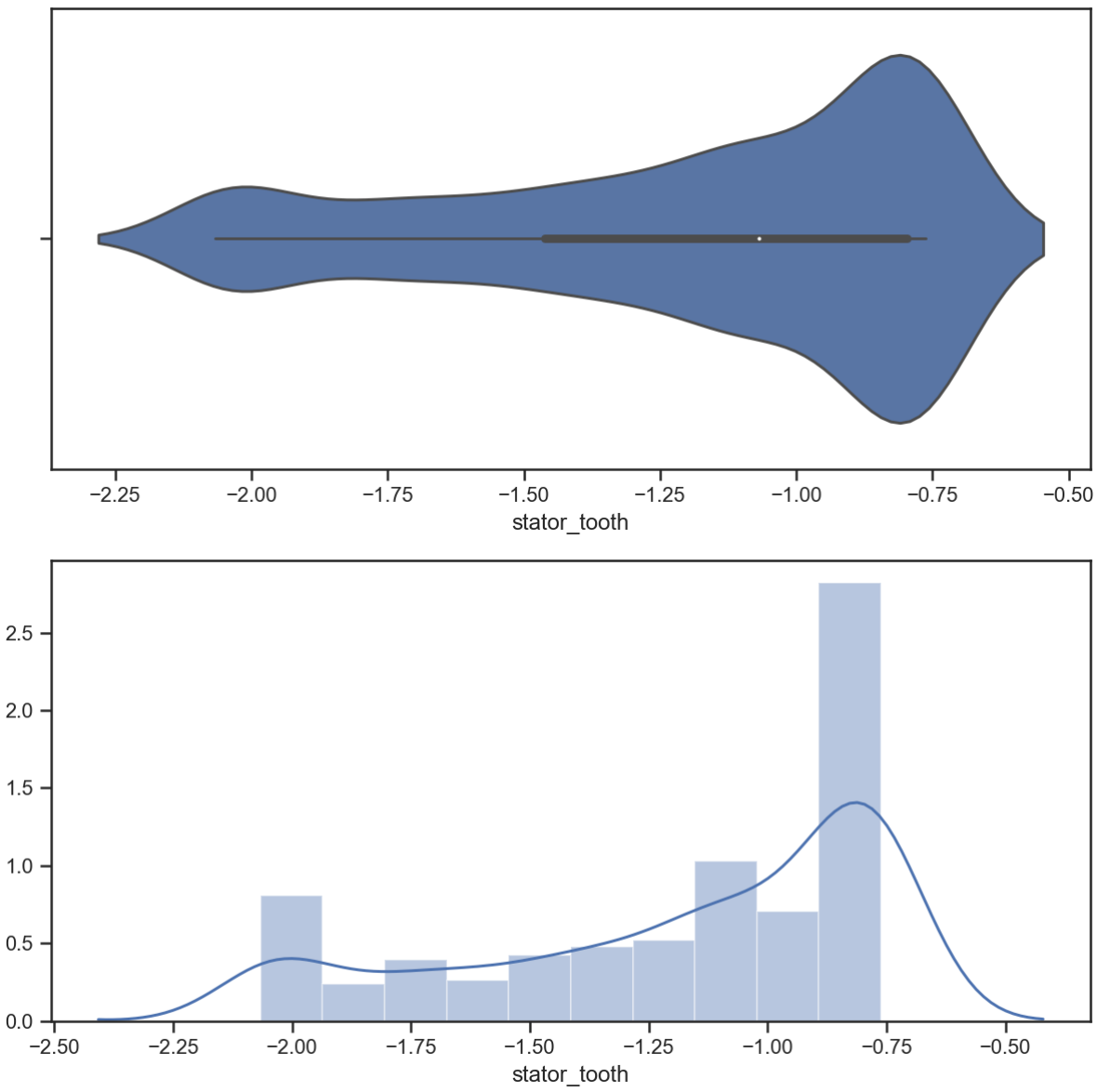
[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x262e29ba6c8>
```



[24]:
```
go.Figure(data=go.Violin(y=data['stator_tooth'], box_visible=True,
                         line_color='black', meanline_visible=True,
                         fillcolor='lightseagreen', opacity=0.6,
                         x0='Stator tooth'))
```

[25]:
```
fig, ax = plt.subplots(2, 1, figsize=(10,10))
sns.violinplot(ax=ax[0], x=data['stator_tooth'])
sns.distplot(data['stator_tooth'], ax=ax[1])
```

[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x262ea3af7c8>
```



Из приведенных графиков видно, что violinplot действительно показывает распределение плотности.

3.4. Информация о корреляции признаков

Проверка корреляции признаков позволяет решить две задачи:

1. Понять какие признаки (колонки датасета) наиболее сильно коррелируют с целевым признаком (в нашем примере это колонка “ambient”). Именно эти признаки будут наиболее информативными для моделей машинного обучения. Признаки, которые слабо коррелируют с целевым признаком, можно попробовать исключить из построения модели, иногда

это повышает качество модели. Нужно отметить, что некоторые алгоритмы машинного обучения автоматически определяют ценность того или иного признака для построения модели.

2. Понять какие нецелевые признаки линейно зависимы между собой. Линейно зависимые признаки, как правило, очень плохо влияют на качество моделей. Поэтому если несколько признаков линейно зависимы, то для построения модели из них выбирают какой-то один признак.

[26]:
```
data.corr()
```

[26]:

	ambient	coolant	u_d	u_q	motor_speed
ambient	1.000000	-0.427510	0.524424	-0.289097	-0.471492
coolant	-0.427510	1.000000	-0.548036	0.387209	0.515636
u_d	0.524424	-0.548036	1.000000	-0.847401	-0.989734
u_q	-0.289097	0.387209	-0.847401	1.000000	0.894330
motor_speed	-0.471492	0.515636	-0.989734	0.894330	1.000000
torque	-0.402339	0.271950	-0.326685	0.058759	0.316890
i_d	0.469157	-0.501213	0.979203	-0.824031	-0.985323
i_q	0.469518	-0.494026	0.878238	-0.579548	-0.866105
pm	-0.639984	0.484276	-0.498522	0.279103	0.410846
stator_yoke	-0.640703	0.556484	-0.496643	0.277906	0.409038
stator_tooth	-0.660257	0.621799	-0.557183	0.319835	0.472693
stator_winding	-0.669521	0.619425	-0.567853	0.327519	0.483656

	torque	i_d	i_q	pm	stator_yoke
ambient	-0.402339	0.469157	0.469518	-0.639984	-0.640703
coolant	0.271950	-0.501213	-0.494026	0.484276	0.556484
u_d	-0.326685	0.979203	0.878238	-0.498522	-0.496643
u_q	0.058759	-0.824031	-0.579548	0.279103	0.277906
motor_speed	0.316890	-0.985323	-0.866105	0.410846	0.409038
torque	1.000000	-0.410614	-0.410365	0.290799	0.290809
i_d	-0.410614	1.000000	0.908954	-0.361008	-0.359227
i_q	-0.410365	0.908954	1.000000	-0.310328	-0.311331
pm	0.290799	-0.361008	-0.310328	1.000000	0.993136
stator_yoke	0.290809	-0.359227	-0.311331	0.993136	1.000000
stator_tooth	0.322963	-0.428183	-0.392604	0.981117	0.990516
stator_winding	0.330354	-0.439822	-0.403707	0.982007	0.989204

	stator_tooth	stator_winding
ambient	-0.660257	-0.669521
coolant	0.621799	0.619425
u_d	-0.557183	-0.567853
u_q	0.319835	0.327519
motor_speed	0.472693	0.483656
torque	0.322963	0.330354
i_d	-0.428183	-0.439822
i_q	-0.392604	-0.403707
pm	0.981117	0.982007
stator_yoke	0.990516	0.989204
stator_tooth	1.000000	0.998447

```
stator_winding      0.998447        1.000000
```

Корреляционная матрица содержит коэффициенты корреляции между всеми парами признаков.

Корреляционная матрица симметрична относительно главной диагонали. На главной диагонали расположены единицы (корреляция признака самого с собой).

На основе корреляционной матрицы можно сделать следующие выводы:

- Целевой признак наиболее сильно коррелирует с температурой ротора (-0.64), температурой ярма статора (-0,64), температурой зубца статора (-0,66) и температурой обмотки статора (-0,67). Эти признаки обязательно следует оставить в модели.
- Целевой признак отчасти коррелирует с напряжением d-компонента (0.52). Этот признак стоит также оставить в модели.
- Целевой признак слабо коррелирует с напряжением q-компонента (-0.29). Скорее всего этот признак стоит исключить из модели, возможно он только ухудшит качество модели.
- Напряжение d-компонента, скорость двигателя и сила тока d-компонента очень сильно коррелируют между собой. Поэтому из этих признаков в модели можно оставлять только один. Также сильно коррелирруют между собой температура ротора, температура ярма статора, температура зубца статора и температура обмотки статора. Поэтому также из этих признаков в модели можно оставлять только один.
- Также можно сделать вывод, что выбирая из признаков напряжение d-компонента, скорость двигателя и сила тока d-компонента лучше выбрать напряжение d-компонента, потому что он сильнее коррелирован с целевым признаком. Если линейно зависимые признаки сильно коррелированы с целевым, то оставляют именно тот признак, который коррелирован с целевым сильнее. Аналогично выбирая из признаков температура ротора, температура ярма статора, температура зубца статора и температура обмотки статора лучше выбрать температура обмотки статора. Однако это далеко не очевидный ответ, поскольку значения корреляции данных признаков довольно одинаковы.

Описание метода corr - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.

По умолчанию при построении матрицы используется коэффициент корреляции Пирсона. Возможно также построить корреляционную матрицу на основе коэффициентов корреляции Кендалла и Спирмена. На практике три метода редко дают значимые различия.

[27]:
```
data.corr(method='pearson')
```

[27]:
```
                  ambient   coolant       u_d       u_q  motor_speed  \
ambient          1.000000 -0.427510  0.524424 -0.289097    -0.471492
coolant         -0.427510  1.000000 -0.548036  0.387209     0.515636
u_d              0.524424 -0.548036  1.000000 -0.847401    -0.989734
u_q             -0.289097  0.387209 -0.847401  1.000000     0.894330
motor_speed     -0.471492  0.515636 -0.989734  0.894330     1.000000
torque          -0.402339  0.271950 -0.326685  0.058759     0.316890
i_d              0.469157 -0.501213  0.979203 -0.824031    -0.985323
i_q              0.469518 -0.494026  0.878238 -0.579548    -0.866105
pm              -0.639984  0.484276 -0.498522  0.279103     0.410846
stator_yoke     -0.640703  0.556484 -0.496643  0.277906     0.409038
stator_tooth    -0.660257  0.621799 -0.557183  0.319835     0.472693
stator_winding  -0.669521  0.619425 -0.567853  0.327519     0.483656

                   torque       i_d       i_q        pm  stator_yoke  \
ambient         -0.402339  0.469157  0.469518 -0.639984    -0.640703
```

```
coolant          0.271950 -0.501213 -0.494026  0.484276     0.556484
u_d             -0.326685  0.979203  0.878238 -0.498522    -0.496643
u_q              0.058759 -0.824031 -0.579548  0.279103     0.277906
motor_speed      0.316890 -0.985323 -0.866105  0.410846     0.409038
torque           1.000000 -0.410614 -0.410365  0.290799     0.290809
i_d             -0.410614  1.000000  0.908954 -0.361008    -0.359227
i_q             -0.410365  0.908954  1.000000 -0.310328    -0.311331
pm               0.290799 -0.361008 -0.310328  1.000000     0.993136
stator_yoke      0.290809 -0.359227 -0.311331  0.993136     1.000000
stator_tooth     0.322963 -0.428183 -0.392604  0.981117     0.990516
stator_winding   0.330354 -0.439822 -0.403707  0.982007     0.989204

                 stator_tooth  stator_winding
ambient             -0.660257       -0.669521
coolant              0.621799        0.619425
u_d                 -0.557183       -0.567853
u_q                  0.319835        0.327519
motor_speed          0.472693        0.483656
torque               0.322963        0.330354
i_d                 -0.428183       -0.439822
i_q                 -0.392604       -0.403707
pm                   0.981117        0.982007
stator_yoke          0.990516        0.989204
stator_tooth         1.000000        0.998447
stator_winding       0.998447        1.000000
```

[28]:
```
data.corr(method='kendall')
```

[28]:
```
                  ambient   coolant       u_d       u_q  motor_speed  \
ambient          1.000000 -0.209727  0.407650 -0.260935    -0.188685
coolant         -0.209727  1.000000 -0.199440  0.065764     0.220941
u_d              0.407650 -0.199440  1.000000 -0.610909    -0.220315
u_q             -0.260935  0.065764 -0.610909  1.000000     0.052480
motor_speed     -0.188685  0.220941 -0.220315  0.052480     1.000000
torque          -0.280045  0.123662 -0.448914  0.313783     0.090421
i_d             -0.174309 -0.075470 -0.510706  0.588319    -0.064194
i_q             -0.127565 -0.078383 -0.507224  0.699235    -0.055746
pm              -0.414531  0.182136 -0.751736  0.550950     0.200506
stator_yoke     -0.415761  0.184060 -0.752873  0.551844     0.201875
stator_tooth    -0.416339  0.188971 -0.756810  0.550478     0.204863
stator_winding  -0.416471  0.183532 -0.750850  0.549751     0.200659

                    torque       i_d       i_q        pm  stator_yoke  \
ambient          -0.280045 -0.174309 -0.127565 -0.414531    -0.415761
coolant           0.123662 -0.075470 -0.078383  0.182136     0.184060
u_d              -0.448914 -0.510706 -0.507224 -0.751736    -0.752873
u_q               0.313783  0.588319  0.699235  0.550950     0.551844
motor_speed       0.090421 -0.064194 -0.055746  0.200506     0.201875
torque            1.000000  0.393556  0.317608  0.454069     0.453999
i_d               0.393556  1.000000  0.653607  0.716330     0.715664
```

```
i_q             0.317608  0.653607  1.000000  0.386420     0.386624
pm              0.454069  0.716330  0.386420  1.000000     0.993427
stator_yoke     0.453999  0.715664  0.386624  0.993427     1.000000
stator_tooth    0.456025  0.707833  0.385463  0.985671     0.985541
stator_winding  0.454500  0.714895  0.385445  0.991444     0.991756

                stator_tooth  stator_winding
ambient            -0.416339       -0.416471
coolant             0.188971        0.183532
u_d                -0.756810       -0.750850
u_q                 0.550478        0.549751
motor_speed         0.204863        0.200659
torque              0.456025        0.454500
i_d                 0.707833        0.714895
i_q                 0.385463        0.385445
pm                  0.985671        0.991444
stator_yoke         0.985541        0.991756
stator_tooth        1.000000        0.985047
stator_winding      0.985047        1.000000
```

[29]:
```
data.corr(method='spearman')
```

[29]:
```
                 ambient   coolant       u_d       u_q  motor_speed  \
ambient         1.000000 -0.307949  0.581245 -0.358748    -0.271968
coolant        -0.307949  1.000000 -0.320865  0.118889     0.307540
u_d             0.581245 -0.320865  1.000000 -0.717390    -0.307292
u_q            -0.358748  0.118889 -0.717390  1.000000     0.071780
motor_speed    -0.271968  0.307540 -0.307292  0.071780     1.000000
torque         -0.400698  0.188662 -0.601255  0.422711     0.124523
i_d            -0.233720 -0.091059 -0.553990  0.703257    -0.095377
i_q            -0.166334 -0.098049 -0.550450  0.782210    -0.090739
pm             -0.590508  0.284642 -0.917172  0.669256     0.283043
stator_yoke    -0.590918  0.285052 -0.917158  0.671001     0.283335
stator_tooth   -0.589106  0.287287 -0.916563  0.665526     0.285679
stator_winding -0.591908  0.285086 -0.916663  0.669131     0.283082

                   torque       i_d       i_q        pm  stator_yoke  \
ambient         -0.400698 -0.233720 -0.166334 -0.590508    -0.590918
coolant          0.188662 -0.091059 -0.098049  0.284642     0.285052
u_d             -0.601255 -0.553990 -0.550450 -0.917172    -0.917158
u_q              0.422711  0.703257  0.782210  0.669256     0.671001
motor_speed      0.124523 -0.095377 -0.090739  0.283043     0.283335
torque           1.000000  0.525548  0.432952  0.610901     0.609855
i_d              0.525548  1.000000  0.841060  0.625337     0.624841
i_q              0.432952  0.841060  1.000000  0.453808     0.453524
pm               0.610901  0.625337  0.453808  1.000000     0.999864
stator_yoke      0.609855  0.624841  0.453524  0.999864     1.000000
stator_tooth     0.608765  0.622959  0.450161  0.998531     0.998491
stator_winding   0.611732  0.624766  0.452894  0.999819     0.999807
```

	stator_tooth	stator_winding
ambient	-0.589106	-0.591908
coolant	0.287287	0.285086
u_d	-0.916563	-0.916663
u_q	0.665526	0.669131
motor_speed	0.285679	0.283082
torque	0.608765	0.611732
i_d	0.622959	0.624766
i_q	0.450161	0.452894
pm	0.998531	0.999819
stator_yoke	0.998491	0.999807
stator_tooth	1.000000	0.998498
stator_winding	0.998498	1.000000

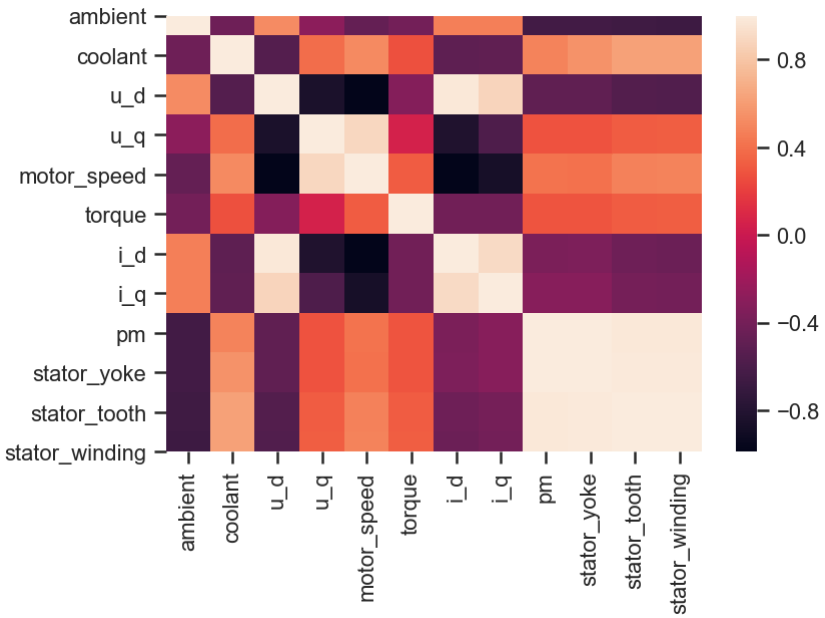
В случае большого количества признаков анализ числовой корреляционной матрицы становится неудобен.

Для визуализации корреляционной матрицы будем использовать “тепловую карту” heatmap которая показывает степень корреляции различными цветами.

Используем метод heatmap библиотеки seaborn - https://seaborn.pydata.org/generated/seaborn.heatma

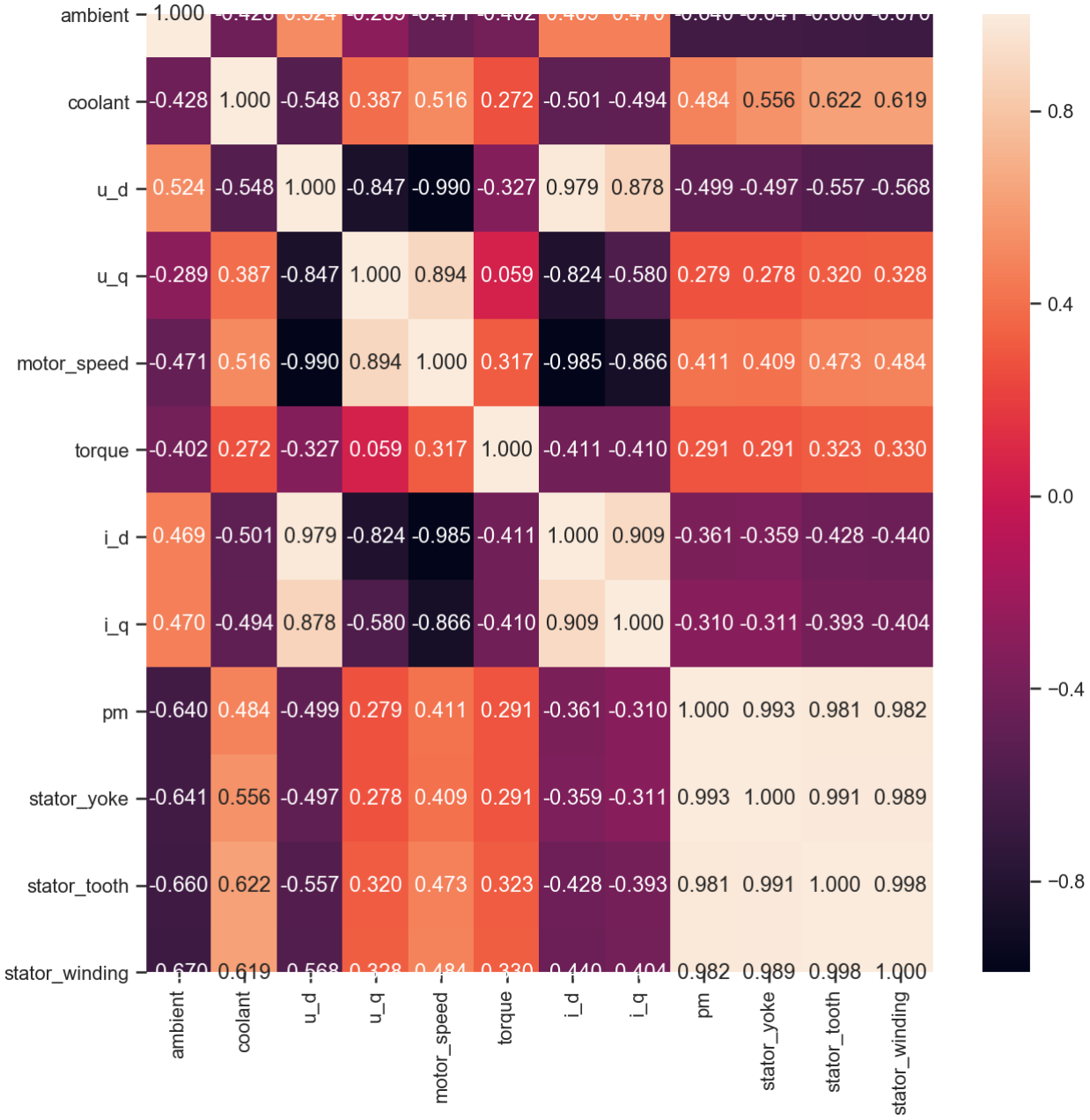
[30]:
```
sns.heatmap(data.corr())
```

[30]: <matplotlib.axes._subplots.AxesSubplot at 0x262e9dc6dc8>



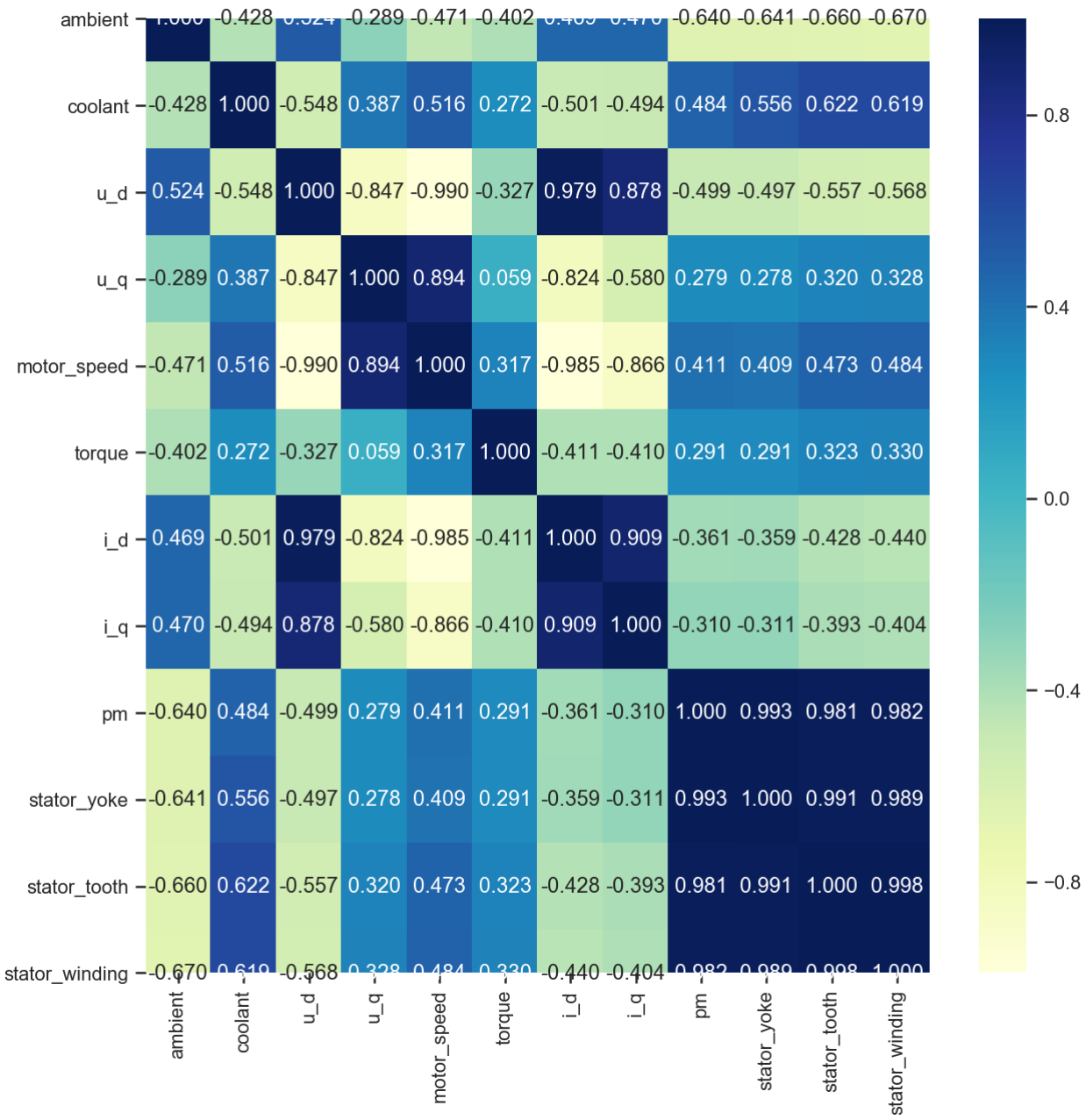
[31]:
```
# Вывод значений в ячейках
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(), annot=True, fmt='.3f', ax=ax)
```

[31]: <matplotlib.axes._subplots.AxesSubplot at 0x262ea9e4548>



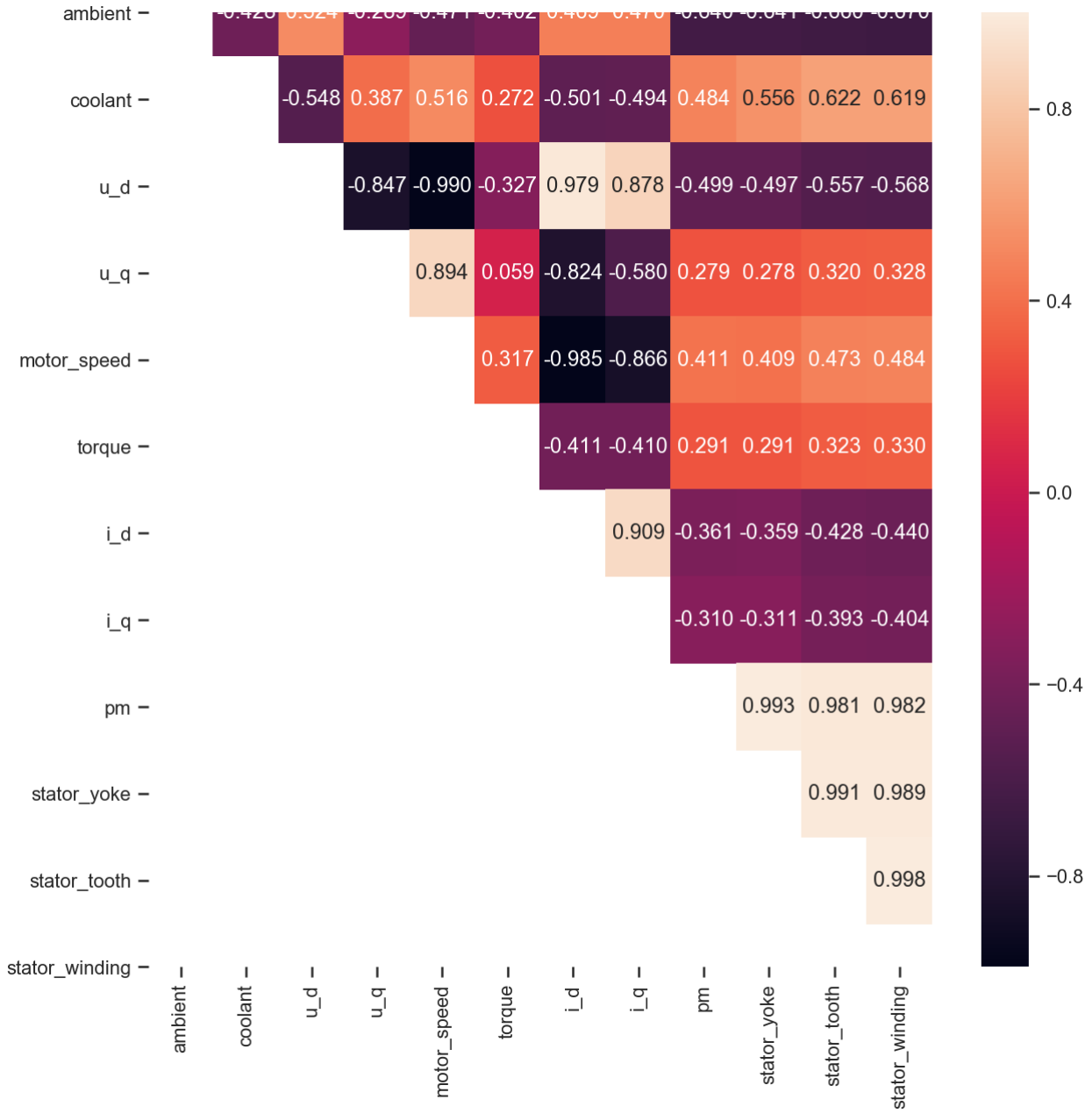
[32]:
```
# Изменение цветовой гаммы
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(), cmap='YlGnBu', annot=True, fmt='.3f', ax=ax)
```

[32]: <matplotlib.axes._subplots.AxesSubplot at 0x262ea9fabc8>



[33]:
```
# Треугольный вариант матрицы
mask = np.zeros_like(data.corr(), dtype=np.bool)
# чтобы оставить нижнюю часть матрицы
# mask[np.triu_indices_from(mask)] = True
# чтобы оставить верхнюю часть матрицы
mask[np.tril_indices_from(mask)] = True
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data.corr(), mask=mask, annot=True, fmt='.3f', ax=ax)
```

[33]: <matplotlib.axes._subplots.AxesSubplot at 0x262eb041808>



[34]:
```
fig, ax = plt.subplots(3, 1, sharex='col', sharey='row',
                       figsize=(15,15))
sns.heatmap(data.corr(method='pearson'), ax=ax[0], annot=True,
            fmt='.2f')
sns.heatmap(data.corr(method='kendall'), ax=ax[1], annot=True,
            fmt='.2f')
sns.heatmap(data.corr(method='spearman'), ax=ax[2], annot=True,
            fmt='.2f')
fig.suptitle('Корреляционные матрицы, построенные различными методами')
ax[0].title.set_text('Pearson')
ax[1].title.set_text('Kendall')
ax[2].title.set_text('Spearman')
```

Корреляционные матрицы, построенные различными методами

Необходимо отметить, что тепловая карта не очень хорошо подходит для определения корреляции нецелевых признаков между собой.

В примере тепловая карта помогает определить значимую корреляцию между признаками pm, stator_yoke, stator_tooth и stator_winding, следовательно только один из этих признаков можно включать в модель.

Но в реальной модели могут быть сотни признаков и коррелирующие признаки могут образовывать группы, состоящие более чем из двух признаков. Увидеть такие группы с помощью тепловой карты сложно.

Для решения задачи предлагается новый вариант визуализации - “Солнечная корреляционная карта” Solar correlation map.

К сожалению, данная библиотека пока работает только через файловый интерфейс и не предназначена для встраивания в ноутбук.

Примеры статей с описанием работы библиотеки:

- https://www.oreilly.com/learning/a-new-visualization-to-beautifully-explore-correlations
- https://www.mtab.com/the-puzzle-of-visualizing-correlations/

Список литературы

[1] Гапанюк Ю. Е. Лабораторная работа «Разведочный анализ данных. Исследование и визуализация данных» [Электронный ресурс] // GitHub. — 2020. — Режим доступа: `https://github.com/ugapanyuk/ml_course_2020/wiki/LAB_MMO__EDA_VISUALIZATION` (дата обращения: 27.02.2020).

[2] Kirgsn. Electric Motor Temperature [Electronic resource] // Kaggle. — 2019. — Access mode: `https://www.kaggle.com/wkirgsn/electric-motor-temperature` (online; accessed: 27.02.2020).

[3] Team The IPython Development. IPython 7.13.0 Documentation [Electronic resource] // Read the Docs. — 2020. — Access mode: `https://ipython.readthedocs.io/en/stable/` (online; accessed: 27.02.2020).

[4] Waskom M. seaborn 0.10.0 documentation [Electronic resource] // PyData. — 2020. — Access mode: `https://seaborn.pydata.org/` (online; accessed: 27.02.2020).

[5] pandas 1.0.1 documentation [Electronic resource] // PyData. — 2020. — Access mode: `http://pandas.pydata.org/pandas-docs/stable/` (online; accessed: 27.02.2020).