

Дисциплина: Методы и технологии машинного обучения

Уровень подготовки: бакалавриат

Направление подготовки: 01.03.02 Прикладная математика и информатика

Семестр: осень 2022/2023

Лабораторная работа №3: Линейные модели. Кросс-валидация.

В практических примерах ниже показано:

- как пользоваться инструментами предварительного анализа для поиска линейных взаимосвязей
- как строить и интерпретировать линейные модели с логарифмами
- как оценивать точность моделей с перекрёстной проверкой (LOOCV, проверка по блокам)

Модели: множественная линейная регрессия Данные: `insurance` (источник:

<https://www.kaggle.com/mirichoi0218/insurance/version/1>)

In [1]:

```
# настройка ширины страницы блокнота .....
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:80% !important; }</style>"))
```

Указания к выполнению

Загружаем пакеты

In [2]:

```
# загрузка пакетов: инструменты -----
# работа с массивами
import numpy as np
# фреймы данных
import pandas as pd
# графики
import matplotlib as mpl
# стили и шаблоны графиков на основе matplotlib
import seaborn as sns
# перекодировка категориальных переменных
from sklearn.preprocessing import LabelEncoder
# тест Шапиро-Уилка на нормальность распределения
from scipy.stats import shapiro
# для таймера
import time

# загрузка пакетов: модели -----
# линейные модели
import sklearn.linear_model as skl_lm
# расчёт MSE
from sklearn.metrics import mean_squared_error
```

```
# кросс-валидация
from sklearn.model_selection import train_test_split, LeaveOneOut
from sklearn.model_selection import KFold, cross_val_score
```

```
In [3]: # константы
# ядро для генератора случайных чисел
my_seed = 9212
# создаём псевдоним для короткого обращения к графикам
plt = mpl.pyplot
# настройка стиля и отображения графиков
# примеры стилей и шаблонов графиков:
# http://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html
mpl.style.use('seaborn-whitegrid')
sns.set_palette("Set2")
# раскомментируйте следующую строку, чтобы посмотреть палитру
# sns.color_palette("Set2")
```

Загружаем данные

Набор данных `insurance` в формате `.csv` доступен для загрузки по адресу:

<https://raw.githubusercontent.com/ania607/ML/main/data/insurance.csv>.

Загружаем данные во фрейм и кодируем категориальные переменные.

```
In [4]: # читаем таблицу из файла .csv во фрейм
fileURL = 'https://raw.githubusercontent.com/ania607/ML/main/data/insurance.csv'
DF_raw = pd.read_csv(fileURL)

# выясняем размерность фрейма
print('Число строк и столбцов в наборе данных:\n', DF_raw.shape)
```

Число строк и столбцов в наборе данных:
(1338, 7)

```
In [5]: # первые 5 строк фрейма
DF_raw.head(5)
```

```
Out[5]:
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
In [6]: # типы столбцов фрейма
DF_raw.dtypes
```

```
Out[6]: age          int64
sex          object
```

```
bmi          float64
children     int64
smoker       object
region       object
charges      float64
dtype: object
```

Проверим, нет ли в таблице пропусков.

```
In [7]: # считаем пропуски в каждом столбце
        DF_raw.isna().sum()
```

```
Out[7]: age          0
sex          0
bmi          0
children     0
smoker       0
region       0
charges      0
dtype: int64
```

Пропусков не обнаружено.

```
In [8]: # кодируем категориальные переменные
        # пол
        sex_dict = {'female' : 1, 'male' : 0}
        DF_raw['sexFemale'] = DF_raw.sex.map(sex_dict)

        # курильщик
        yn_dict = {'yes' : 1, 'no' : 0}
        DF_raw['smokerYes'] = DF_raw.smoker.map(yn_dict)

        # находим уникальные регионы
        DF_raw['region'].unique()
```

```
Out[8]: array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)
```

```
In [9]: # добавляем фиктивные на регион: число фиктивных = число уникальных - 1
        df_dummy = pd.get_dummies(DF_raw[['region']], drop_first=True)
        df_dummy.head(5)
```

```
Out[9]:
```

	region_northwest	region_southeast	region_southwest
0	0	0	1
1	0	1	0
2	0	1	0
3	1	0	0
4	1	0	0

```
In [10]: # объединяем с исходным фреймом
        DF_all = pd.concat([DF_raw.reset_index(drop=True), df_dummy], axis=1)

        # сколько теперь столбцов
        DF_all.shape
```

Out[10]: (1338, 12)

```
In [11]: # смотрим первые 8 столбцов
DF_all.iloc[:, :8].head(5)
```

```
Out[11]:
```

	age	sex	bmi	children	smoker	region	charges	sexFemale
0	19	female	27.900	0	yes	southwest	16884.92400	1
1	18	male	33.770	1	no	southeast	1725.55230	0
2	28	male	33.000	3	no	southeast	4449.46200	0
3	33	male	22.705	0	no	northwest	21984.47061	0
4	32	male	28.880	0	no	northwest	3866.85520	0

```
In [12]: # смотрим последние 5 столбцов
DF_all.iloc[:, 8:].head(5)
```

```
Out[12]:
```

	smokerYes	region_northwest	region_southeast	region_southwest
0	1	0	0	1
1	0	0	1	0
2	0	0	1	0
3	0	1	0	0
4	0	1	0	0

```
In [13]: # оставляем в наборе данных только то, что нужно
# (плюс метки регионов для графиков)
DF_all = DF_all[['charges', 'age', 'sexFemale', 'bmi', 'children', 'smokerYes',
                  'region_northwest', 'region_southeast',
                  'region_southwest', 'region']]

# перекодируем регион в числовой фактор,
# чтобы использовать на графиках
class_le = LabelEncoder()
DF_all['region'] = class_le.fit_transform(DF_all['region'].values)

DF_all.columns
```

```
Out[13]: Index(['charges', 'age', 'sexFemale', 'bmi', 'children', 'smokerYes',
                'region_northwest', 'region_southeast', 'region_southwest', 'region'],
                dtype='object')
```

```
In [14]: DF_all.dtypes
```

```
Out[14]: charges          float64
age                int64
sexFemale          int64
bmi                float64
children           int64
```

```
smokerYes          int64
region_northwest    uint8
region_southeast    uint8
region_southwest    uint8
region             int32
dtype: object
```

```
In [15]: # удаляем фрейм-исходник
del DF_raw
```

Прежде чем переходить к анализу данных, разделим фрейм на две части: одна (90%) станет основой для обучения моделей, на вторую (10%) мы сделаем прогноз по лучшей модели.

```
In [16]: # данные для построения моделей
DF = DF_all.sample(frac = 0.9, random_state = my_seed)

# данные для прогнозов
DF_predict = DF_all.drop(DF.index)
```

Предварительный анализ данных

Считаем описательные статистики

Рассчитаем описательные статистики для непрерывных переменных. Из таблицы ниже можно видеть, что переменная `charges`, которая является зависимой переменной модели, сильно отличается по масштабу от всех остальных. Также заметим, что из всех объясняющих только переменная `children` принимает нулевые значения. Остальные показатели положительны.

```
In [17]: # описательные статистики для непрерывных переменных
DF[['charges', 'age', 'bmi', 'children']].describe()
```

```
Out[17]:
```

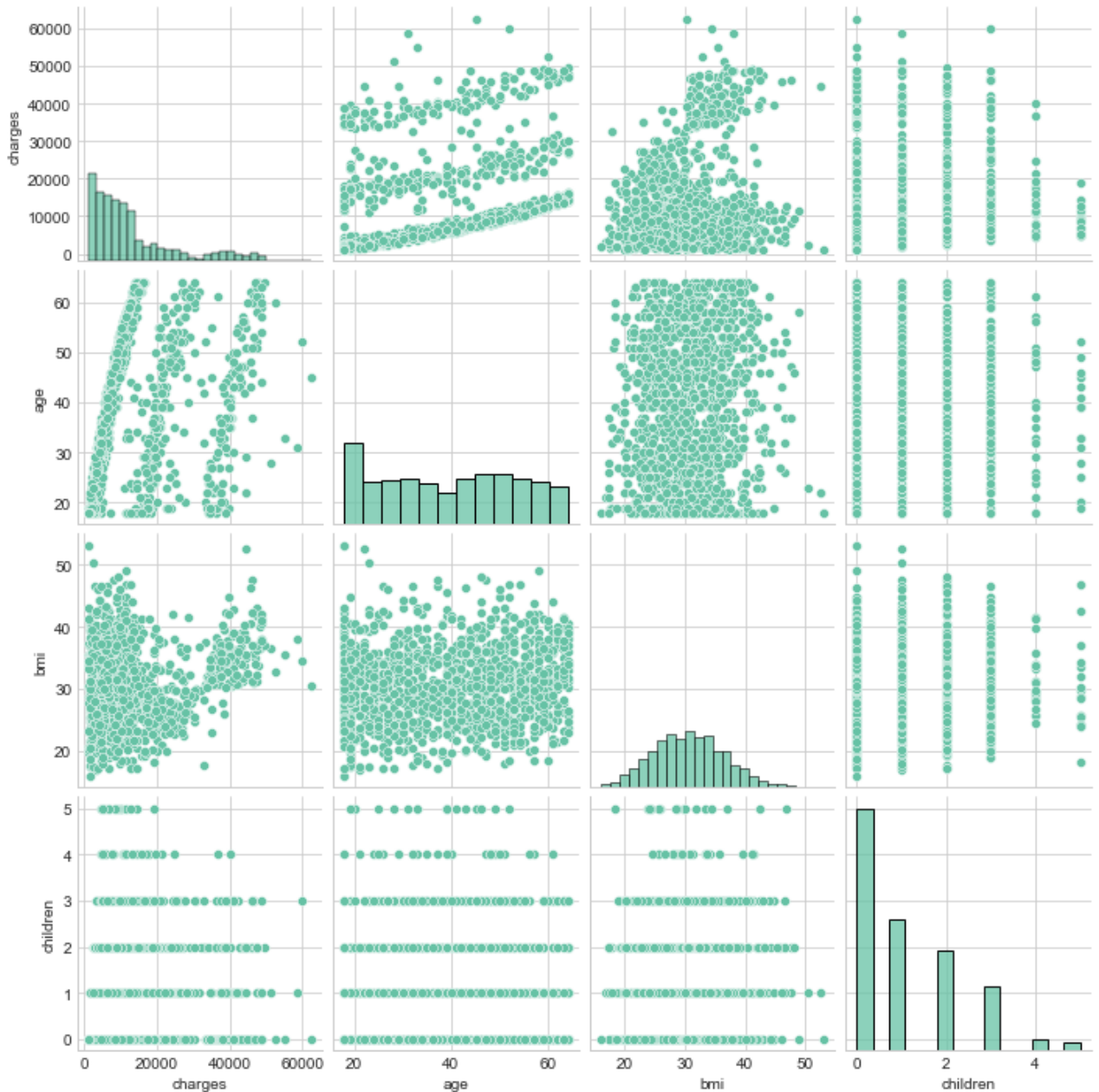
	charges	age	bmi	children
count	1204.000000	1204.000000	1204.000000	1204.000000
mean	13172.801289	39.453488	30.704007	1.094684
std	12019.293394	14.028898	6.054108	1.217497
min	1121.873900	18.000000	15.960000	0.000000
25%	4750.065725	27.000000	26.400000	0.000000
50%	9382.033000	40.000000	30.495000	1.000000
75%	16328.508137	51.000000	34.717500	2.000000
max	62592.873090	64.000000	53.130000	5.000000

Строим графики

Посмотрим на графики взаимного разброса непрерывных переменных.

```
In [18]: # матричный график разброса с линиями регрессии
```

```
sns.pairplot(DF[['charges', 'age', 'bmi', 'children']])
plt.show()
```



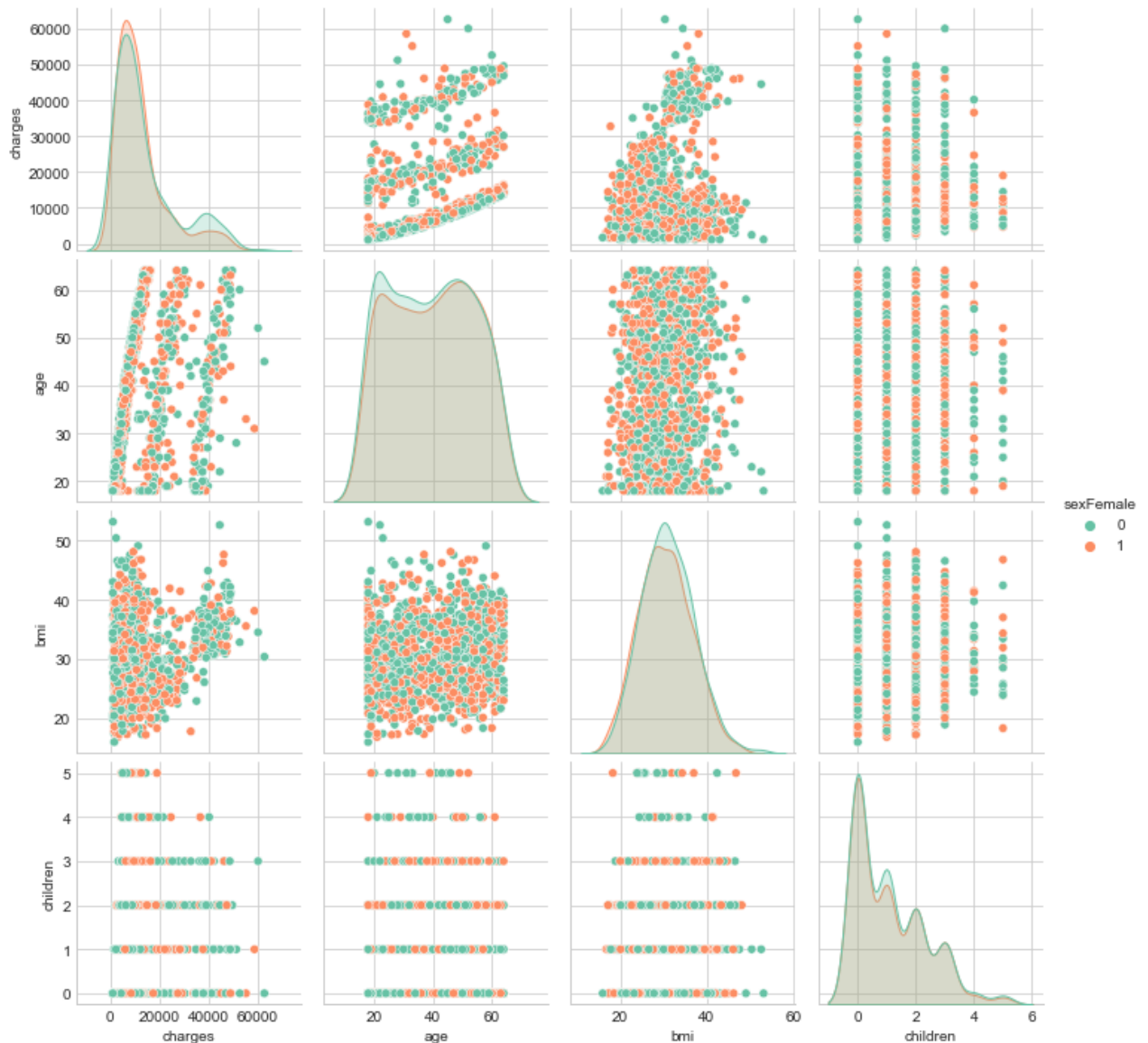
Судя по этим графикам:

- распределение зависимой `charges` не является нормальным;
- из всех объясняющих нормально распределена только `bmi` ;
- имеется три уровня стоимости страховки, что заметно на графиках разброса `charges` от `age` ;
- облако наблюдений на графике `charges` от `bmi` делится на две неравные части;
- объясняющая `children` дискретна, что очевидно из её смысла: количество детей;
- разброс значений `charges` у застрахованных с количеством детей 5 (максимум из таблицы выше) намного меньше, чем у остальных застрахованных.

Наблюдаемые закономерности могут объясняться влиянием одной или нескольких из фиктивных объясняющих переменных. Построим график, раскрасив точки цветом в

зависимости от пола застрахованного лица.

```
In [19]: # матричный график разброса с цветом по полу
sns.pairplot(DF[['charges', 'age', 'bmi', 'children',
                'sexFemale']], hue='sexFemale')
plt.show()
```



Похоже, что оранжевые и зелёные точки распределены по обозначенным выше интересным участкам графиков более-менее равномерно, за исключением столбца значений `children == 5`, где зелёных визуально больше. Проверим это.

```
In [20]: DF.groupby('sexFemale')['children'].value_counts()
```

```
Out[20]: sexFemale  children
0           0           264
           1           151
           2           108
           3            70
           4            13
           5            10
1           0           259
```

1	133
2	108
3	70
4	10
5	8

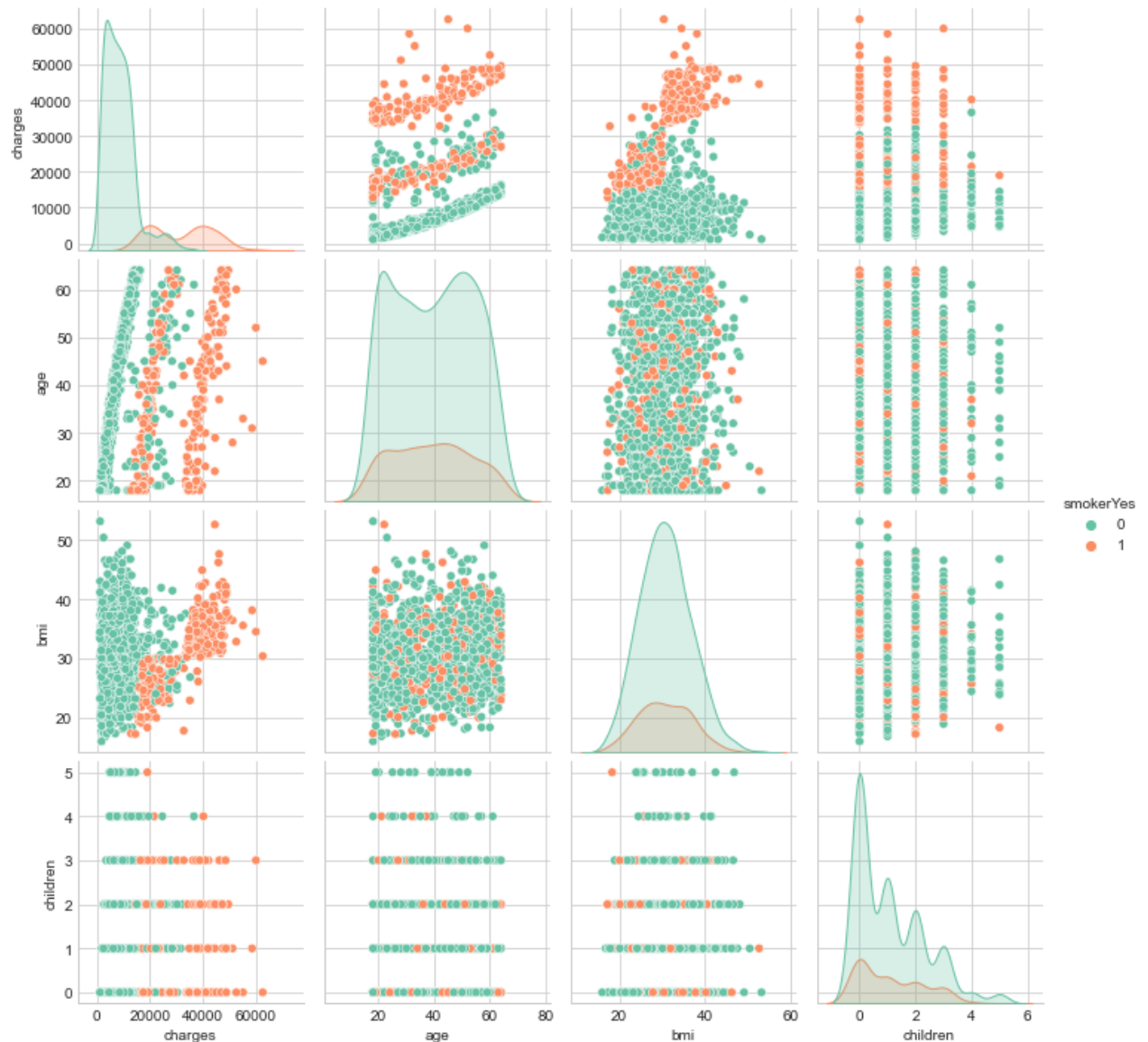
Name: children, dtype: int64

Очевидно, подсчёт частот не подтверждает наше наблюдение: застрахованных мужчин и женщин с одним и тем же количеством детей примерно одинаково.

Теперь покажем цветом на графиках отношение застрахованных лиц к курению.

In [21]:

```
# матричный график разброса с цветом по smokerYes
sns.pairplot(DF[['charges', 'age', 'bmi', 'children',
                'smokerYes']], hue='smokerYes')
plt.show()
```



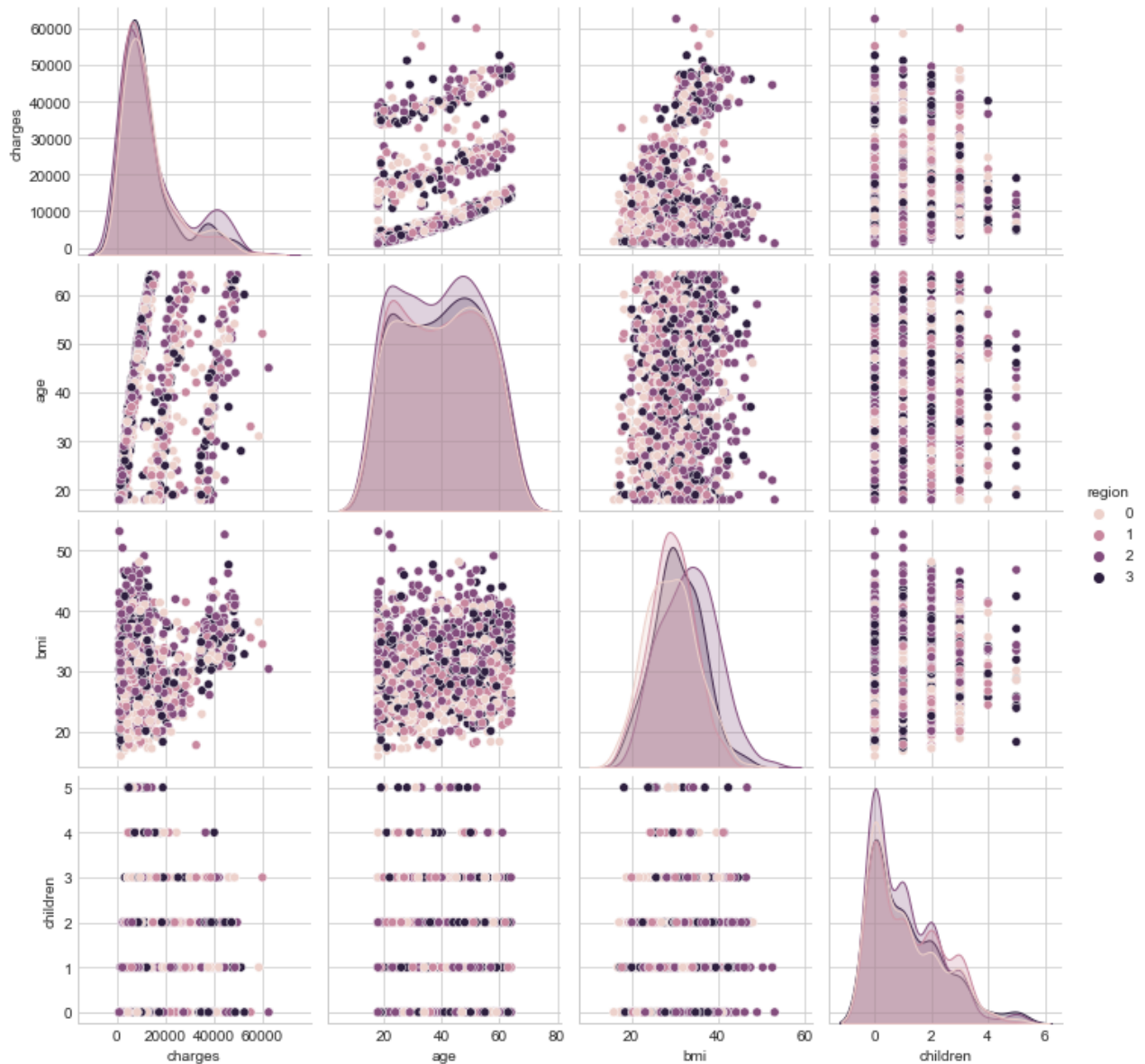
Здесь фиктивная переменная `smokerYes` явно влияет на сумму страховки. Во-первых, для некурящих действует свой тариф в зависимости от возраста (у линейной модели отличается константа). Во-вторых, курильщики с индексом массы тела выше среднего (`bmi`) также платят за медицинскую страховку больше, однако, судя по графику разброса, здесь присутствует

влияние ещё какого-то фактора.

Наконец, покажем с помощью цвета на графиках регионы.

In [22]:

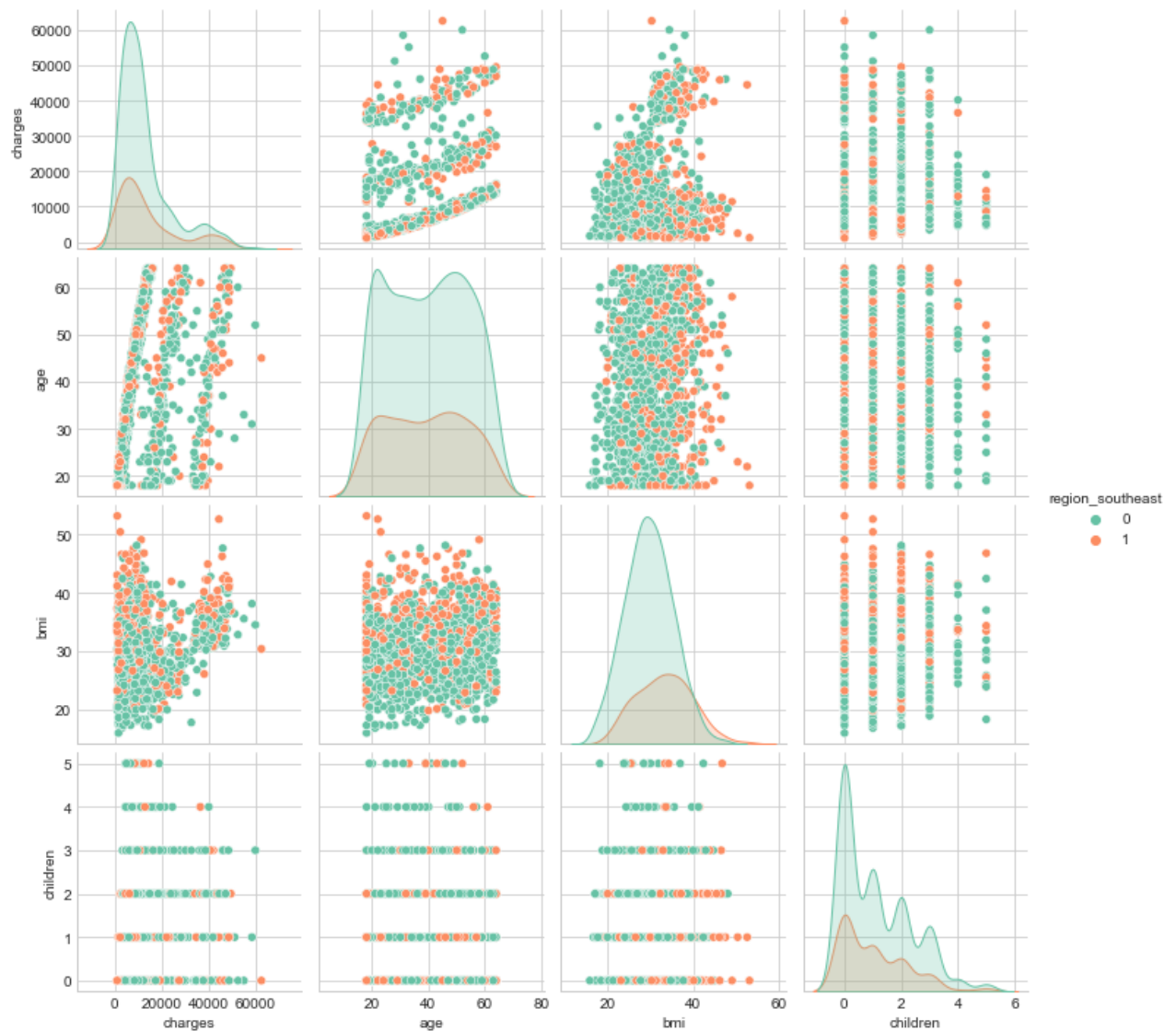
```
# матричный график разброса с цветом по region
sns.pairplot(DF[['charges', 'age', 'bmi', 'children',
                'region']], hue='region')
plt.show()
```



Судя по графикам плотности, переменная `region` не делит непрерывные переменные фрейма на какие-то подгруппы. Отличаются по регионам разве что средние значения `bmi`: похоже, что у региона с кодом 2 (что соответствует `region_southeast == 1`, в чём можно убедиться, заглянув во фрейм данных) средний индекс массы тела немного выше, чем в других регионах. Нарисуем график отдельно по `region_southeast`.

In [23]:

```
# матричный график разброса с цветом по региону southeast
sns.pairplot(DF[['charges', 'age', 'bmi', 'children',
                'region_southeast']], hue='region_southeast')
plt.show()
```



Увы, явных шаблонов, связанных с этим регионом, на графиках разброса не обнаруживается. Посмотрим на корреляционные матрицы непрерывных переменных фрейма.

```
In [24]: # корреляционная матрица по всем наблюдениям
corr_mat = DF[['charges', 'age', 'bmi', 'children']].corr()
corr_mat.style.background_gradient(cmap='coolwarm').set_precision(2)
```

```
Out [24]:
```

	charges	age	bmi	children
charges	1.00	0.31	0.18	0.07
age	0.31	1.00	0.10	0.06
bmi	0.18	0.10	1.00	0.01
children	0.07	0.06	0.01	1.00

Без разбиения на классы наблюдений по фиктивным переменным максимальная теснота линейной взаимосвязи соответствует коэффициенту корреляции 0.30 между `charges` и `age`. Посчитаем корреляционные матрицы для курящих и некурящих застрахованных лиц.

```
In [25]: # корреляционная матрица по классу курильщиков
```

```
corr_mat = DF.loc[DF['smokerYes'] == 1][['charges', 'age',
                                          'bmi', 'children']].corr()
corr_mat.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[25]:

	charges	age	bmi	children
charges	1.00	0.38	0.80	0.00
age	0.38	1.00	0.05	0.07
bmi	0.80	0.05	1.00	-0.06
children	0.00	0.07	-0.06	1.00

In [26]:

```
# корреляционная матрица по классу не курильщиков
corr_mat = DF.loc[DF['smokerYes'] == 0][['charges', 'age',
                                          'bmi', 'children']].corr()
corr_mat.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[26]:

	charges	age	bmi	children
charges	1.00	0.63	0.07	0.16
age	0.63	1.00	0.11	0.06
bmi	0.07	0.11	1.00	0.03
children	0.16	0.06	0.03	1.00

Можно убедиться, что линейные связи с учётом `smokerYes` значительно усиливаются, причём у тесно связанным с `charges` объясняющим добавляется `bmi`.

Логарифмируем зависимую переменную

Важным допущением линейной регрессии является нормальность зависимой переменной. Чтобы добиться нормального распределения, используют логарифмирование либо преобразование Бокса-Кокса. В этой лабораторной остановимся на логарифмировании.

In [27]:

```
# логарифмируем зависимую переменную
DF['log_charges'] = np.log(DF['charges'])

# описательные статистики для непрерывных показателей
DF[['charges', 'log_charges', 'age', 'bmi', 'children']].describe()
```

Out[27]:

	charges	log_charges	age	bmi	children
count	1204.000000	1204.000000	1204.000000	1204.000000	1204.000000
mean	13172.801289	9.092599	39.453488	30.704007	1.094684
std	12019.293394	0.919331	14.028898	6.054108	1.217497
min	1121.873900	7.022756	18.000000	15.960000	0.000000
25%	4750.065725	8.465914	27.000000	26.400000	0.000000
50%	9382.033000	9.146552	40.000000	30.495000	1.000000

	charges	log_charges	age	bmi	children
75%	16328.508137	9.700663	51.000000	34.717500	2.000000
max	62592.873090	11.044407	64.000000	53.130000	5.000000

Отметим, как сильно логарифмирование повлияло на масштаб значений зависимой переменной. Проведём формальные тесты на нормальность.

In [28]:

```
# тестируем на нормальность
for col in ['charges', 'log_charges']:
    stat, p = shapiro(DF[col])
    print(col, 'Statistics=%.2f, p=%.4f' % (stat, p))
    # интерпретация
    alpha = 0.05
    if p > alpha:
        print('Распределение нормально (H0 не отклоняется)\n')
    else:
        print('Распределение не нормально (H0 отклоняется)\n')
```

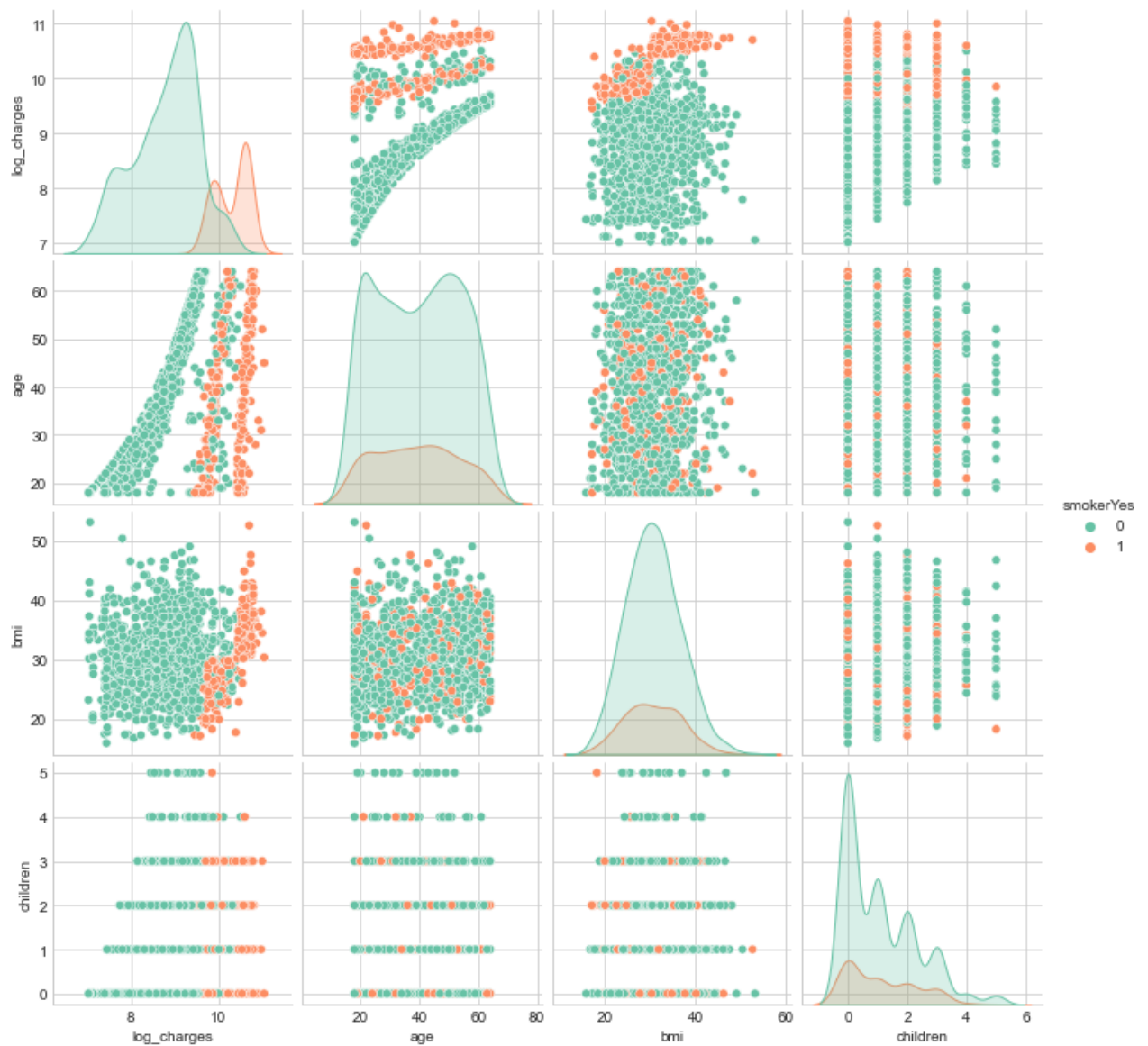
```
charges Statistics=0.81, p=0.0000
Распределение не нормально (H0 отклоняется)
```

```
log_charges Statistics=0.98, p=0.0000
Распределение не нормально (H0 отклоняется)
```

Тест Шапиро-Уилка показывает, что после логарифмирования `log_charges` по-прежнему не распределена нормально, однако функция плотности на графике ниже выглядит более нормальной, чем была у `charges`. Логарифмирование меняет взаимосвязи между переменными, и, судя по графикам разброса, в нашем случае некоторые стали нелинейными. Тем не менее, корреляционные матрицы имеют ту же структуру.

In [29]:

```
# матричный график разброса с цветом по smokerYes
sns.pairplot(DF[['log_charges', 'age', 'bmi', 'children',
                 'smokerYes']], hue='smokerYes')
plt.show()
```



```
In [30]: # корреляционная матрица по классу не курильщиков
corr_mat = DF.loc[DF['smokerYes'] == 0][['log_charges', 'age',
                                          'bmi', 'children']].corr()
corr_mat.style.background_gradient(cmap='coolwarm').set_precision(2)
```

```
Out[30]:
```

	log_charges	age	bmi	children
log_charges	1.00	0.80	0.08	0.26
age	0.80	1.00	0.11	0.06
bmi	0.08	0.11	1.00	0.03
children	0.26	0.06	0.03	1.00

```
In [31]: # корреляционная матрица по классу курильщиков
corr_mat = DF.loc[DF['smokerYes'] == 1][['log_charges', 'age',
                                          'bmi', 'children']].corr()
corr_mat.style.background_gradient(cmap='coolwarm').set_precision(2)
```

```
Out[31]:
```

	log_charges	age	bmi	children
--	-------------	-----	-----	----------

	log_charges	age	bmi	children
log_charges	1.00	0.39	0.80	0.01
age	0.39	1.00	0.05	0.07
bmi	0.80	0.05	1.00	-0.06
children	0.01	0.07	-0.06	1.00

Строим модели регрессии

Спецификация моделей

По итогам предварительного анализа данных можно предложить следующие спецификации линейных регрессионных моделей:

1. `fit_lm_1`: $\hat{charges} = \hat{\beta}_0 + \hat{\beta}_1 \cdot smokerYes + \hat{\beta}_2 \cdot age + \hat{\beta}_3 \cdot bmi$
2. `fit_lm_2`: $\hat{charges} = \hat{\beta}_0 + \hat{\beta}_1 \cdot smokerYes + \hat{\beta}_2 \cdot age \cdot smokerYes + \hat{\beta}_3 \cdot bmi$
3. `fit_lm_3`: $\hat{charges} = \hat{\beta}_0 + \hat{\beta}_1 \cdot smokerYes + \hat{\beta}_2 \cdot bmi \cdot smokerYes + \hat{\beta}_3 \cdot age$
4. `fit_lm_4`:
 $\hat{charges} = \hat{\beta}_0 + \hat{\beta}_1 \cdot smokerYes + \hat{\beta}_2 \cdot bmi \cdot smokerYes + \hat{\beta}_3 \cdot age \cdot smokerYes$
5. `fit_lm_1_log`: то же, что `fit_lm_1`, но для зависимой $\log_{\hat{charges}}$
6. `fit_lm_2_log`: то же, что `fit_lm_2`, но для зависимой $\log_{\hat{charges}}$
7. `fit_lm_3_log`: то же, что `fit_lm_3`, но для зависимой $\log_{\hat{charges}}$
8. `fit_lm_4_log`: то же, что `fit_lm_4`, но для зависимой $\log_{\hat{charges}}$

Кроме того, добавим в сравнение модели зависимости `charges` и `log_charges` от всех объясняющих переменных: `fit_lm_0` и `fit_lm_0_log` соответственно.

Обучение и интерпретация

Создаём матрицы значений объясняющих переменных (X) и вектора значений зависимой (y) для всех моделей.

In [32]:

```
# данные для моделей 1, 5
df1 = DF[['charges', 'smokerYes', 'age', 'bmi']]

# данные для моделей 2, 6
df2 = DF[['charges', 'smokerYes', 'age', 'bmi']]
df2.loc[:, 'age_smokerYes'] = df2.loc[:, 'age'] * df2.loc[:, 'smokerYes']
df2 = df2.drop(['age'], axis=1)

# данные для моделей 3, 7
df3 = DF[['charges', 'smokerYes', 'age', 'bmi']]
df3.loc[:, 'bmi_smokerYes'] = df3.loc[:, 'bmi'] * df3.loc[:, 'smokerYes']
df3 = df3.drop(['bmi'], axis=1)

# данные для моделей 4, 8
df4 = DF[['charges', 'smokerYes', 'age', 'bmi']]
```

```
df4.loc[:, 'age_smokerYes'] = df4.loc[:, 'age'] * df4.loc[:, 'smokerYes']
df4.loc[:, 'bmi_smokerYes'] = df4.loc[:, 'bmi'] * df4.loc[:, 'smokerYes']
df4 = df4.drop(['age', 'bmi'], axis=1)

# данные для моделей 9, 10
df0 = DF.drop(['log_charges', 'region'], axis=1)
```

```
C:\Users\user\anaconda3\lib\site-packages\pandas\core\indexing.py:1597: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
self.obj[key] = value
C:\Users\user\anaconda3\lib\site-packages\pandas\core\indexing.py:1676: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
self._setitem_single_column(ilocs[0], value, pi)
```

Построим модели от всех объясняющих переменных на всех наблюдениях `DF`, чтобы проинтерпретировать параметры. В модели для зависимой переменной `charges` интерпретация стандартная:

1. Константа – базовый уровень зависимой переменной, когда все объясняющие равны 0.
2. Коэффициент при объясняющей переменной X показывает, на сколько своих единиц измерения изменится Y , если X увеличится на одну свою единицу измерения.

```
In [33]: lm = skl_lm.LinearRegression()

# модель со всеми объясняющими, y
X = df0.drop(['charges'], axis=1)
y = df0.charges.values.reshape(-1, 1)
fit_lm_0 = lm.fit(X, y)
print('модель fit_lm_0:\n',
      'константа ', np.around(fit_lm_0.intercept_, 3),
      '\n объясняющие ', list(X.columns.values),
      '\n коэффициенты ', np.around(fit_lm_0.coef_, 3))

модель fit_lm_0:
константа [-11888.329]
объясняющие ['age', 'sexFemale', 'bmi', 'children', 'smokerYes', 'region_north
west', 'region_southeast', 'region_southwest']
коэффициенты [[ 259.625  290.528  324.171  446.331 23783.587 -219.521 -9
59.966
-856.272]]
```

```
In [34]: # оценим MSE на обучающей
# прогнозы
y_pred = fit_lm_0.predict(X)
MSE = sum((y - y_pred.reshape(-1, 1))**2) / len(y)
MSE
```

```
Out[34]: array([35968901.65408054])
```


С интерпретацией модели на логарифме Y дела обстоят сложнее:

1. Константу сначала надо экспоненцировать, далее интерпретировать как для обычной модели регрессии.
2. Коэффициент при X нужно экспоненцировать, затем вычесть из получившегося числа 1, затем умножить на 100. Результат показывает, на сколько процентов изменится (увеличится, если коэффициент положительный, и уменьшится, если отрицательный) зависимая переменная, если X увеличится на одну свою единицу измерения.

```
In [35]: # модель со всеми объясняющими, y_log
X = df0.drop(['charges'], axis=1)
y = np.log(df0.charges).values.reshape(-1, 1)
fit_lm_0_log = lm.fit(X, y)
print('модель fit_lm_0_log:\n',
      'константа ', np.around(fit_lm_0_log.intercept_, 3),
      '\n объясняющие ', list(X.columns.values),
      '\n коэффициенты ', np.around(fit_lm_0_log.coef_, 3))

модель fit_lm_0_log:
константа [6.924]
объясняющие ['age', 'sexFemale', 'bmi', 'children', 'smokerYes', 'region_north
west', 'region_southeast', 'region_southwest']
коэффициенты [[ 0.035  0.086  0.013  0.104  1.538 -0.051 -0.149 -0.114]]
```

```
In [36]: # пересчёт константы для интерпретации
np.round(np.exp(fit_lm_0_log.intercept_), 1)
```

```
Out[36]: array([1016.8])
```

```
In [37]: # пересчёт коэффициентов для их интерпретации
np.round((np.exp(fit_lm_0_log.coef_) - 1) * 100, 1)
```

```
Out[37]: array([[ 3.6,   8.9,   1.3,  11. , 365.6,  -4.9, -13.8, -10.8]])
```

```
In [38]: # оценим MSE на обучающей
# прогнозы
y_pred = fit_lm_0_log.predict(X)
MSE_log = sum((np.exp(y) - np.exp(y_pred).reshape(-1, 1))**2) / len(y)
MSE_log
```

```
Out[38]: array([68062396.99291997])
```

```
In [39]: print('MSE_train модели для charges меньше MSE_train',
      'модели для log(charges) в ', np.around(MSE_log / MSE, 1), 'раз')
```

```
MSE_train модели для charges меньше MSE_train модели для log(charges) в [1.9] раз
```

Оценка точности

LOOCV

Сделаем перекрёстную проверку точности моделей по одному наблюдению.

```
In [40]: # LeaveOneOut CV
loo = LeaveOneOut()

# модели для y
scores = list()
# таймер
tic = time.perf_counter()
for df in [df0, df1, df2, df3, df4] :
    loo.get_n_splits(df)
    X = df.drop(['charges'], axis=1)
    y = df.charges
    score = cross_val_score(lm, X, y, cv=loo,
                           scoring='neg_mean_squared_error').mean()
    scores.append(score)

# таймер
toc = time.perf_counter()
print(f"Расчёты методом LOOCV заняли {toc - tic:0.2f} секунд")
```

Расчёты методом LOOCV заняли 19.40 секунд

```
In [41]: # модели для y_log
scores_log = list()
# таймер
tic = time.perf_counter()
for df in [df0, df1, df2, df3, df4] :
    loo.get_n_splits(df)
    X = df.drop(['charges'], axis=1)
    y = np.log(df.charges)
    score = cross_val_score(lm, X, y, cv=loo, n_jobs=1,
                           scoring='neg_mean_squared_error').mean()
    scores_log.append(score)

# таймер
toc = time.perf_counter()
print(f"Расчёты методом LOOCV заняли {toc - tic:0.2f} секунд")
```

Расчёты методом LOOCV заняли 18.75 секунд

Сравним ошибки для моделей на исходных значениях `charges` с ошибками моделей на логарифме.

```
In [42]: [np.around(-x, 2) for x in scores]
```

```
Out[42]: [36576879.42, 36751828.66, 46613497.18, 23782780.05, 34990547.45]
```

```
In [43]: [np.around(-x, 3) for x in scores_log]
```

```
Out[43]: [0.197, 0.217, 0.461, 0.204, 0.455]
```

Очевидно, что между собой эти цифры сравнивать нельзя, поскольку процедура `cross_val_score()` при расчёте MSE не делает экспоненцирования Y . Поэтому, если требуется выбрать из всех моделей лучшую, схема работы должна быть следующей:

1. Найти наилучшую модель для Y .
2. Найти наилучшую модель для $\log(Y)$.
3. Перестроить их без перекрёстной проверки, оценив MSE методом проверочной выборки.
4. Рассчитать MSE перестроенных моделей вручную и сравнить.

```
In [44]: # самая точная на charges
fits = ['fit_lm_0', 'fit_lm_1', 'fit_lm_2', 'fit_lm_3', 'fit_lm_4']
print('Наименьшая ошибка на тестовой с LOOCV у модели',
      fits[scores.index(max(scores))],
      ':\nMSE_loocv =', np.around(-max(scores), 0))
```

Наименьшая ошибка на тестовой с LOOCV у модели fit_lm_3 :
MSE_loocv = 23782780.0

```
In [45]: # самая точная на log(charges)
fits = ['fit_lm_0_log', 'fit_lm_1_log', 'fit_lm_2_log',
        'fit_lm_3_log', 'fit_lm_4_log']
print('Наименьшая ошибка на тестовой с LOOCV у модели',
      fits[scores_log.index(max(scores_log))],
      ':\nMSE_loocv =', np.around(-max(scores_log), 3))
```

Наименьшая ошибка на тестовой с LOOCV у модели fit_lm_0_log :
MSE_loocv = 0.197

Перекрёстная проверка по блокам

Теоретически этот метод менее затратен, чем LOOCV. Проверим на наших моделях.

```
In [46]: # Перекрёстная проверка по 10 блокам
folds = 10

# ядра для разбиений перекрёстной проверкой
r_state = np.arange(my_seed, my_seed + 9)

# модели для y
scores = list()
# таймер
tic = time.perf_counter()
i = 0
for df in [df0, df1, df2, df3, df4] :
    X = df.drop(['charges'], axis=1)
    y = df.charges
    kf_10 = KFold(n_splits=folds, random_state=r_state[i],
                  shuffle=True)
    score = cross_val_score(lm, X, y, cv=kf_10,
                             scoring='neg_mean_squared_error').mean()
    scores.append(score)
    i+=1

# таймер
toc = time.perf_counter()
print(f"Расчёты методом CV по 10 блокам заняли {toc - tic:0.2f} секунд")
```

Расчёты методом CV по 10 блокам заняли 0.17 секунд

```
In [47]: # Перекрёстная проверка по 10 блокам
folds = 10
```

```

# ядра для разбиений перекрёстной проверкой
r_state = np.arange(my_seed, my_seed + 9)

# модели для y
scores_log = list()
# таймер
tic = time.perf_counter()
i = 0
for df in [df0, df1, df2, df3, df4] :
    X = df.drop(['charges'], axis=1)
    y = np.log(df.charges)
    kf_10 = KFold(n_splits=folds, random_state=r_state[i],
                  shuffle=True)
    score = cross_val_score(lm, X, y, cv=kf_10,
                            scoring='neg_mean_squared_error').mean()
    scores_log.append(score)
    i+=1

# таймер
toc = time.perf_counter()
print(f"Расчёты методом CV по 10 блокам заняли {toc - tic:0.2f} секунд")

```

Расчёты методом CV по 10 блокам заняли 0.16 секунд

In [48]:

```

# самая точная на charges
fits = ['fit_lm_0', 'fit_lm_1', 'fit_lm_2', 'fit_lm_3', 'fit_lm_4']
print('Наименьшая ошибка на тестовой с k-fold10 у модели',
      fits[scores.index(max(scores))],
      ':\nMSE_kf10 =', np.around(-max(scores), 0))

```

Наименьшая ошибка на тестовой с k-fold10 у модели fit_lm_3 :
MSE_kf10 = 23764311.0

In [49]:

```

# самая точная на log(charges)
fits = ['fit_lm_0_log', 'fit_lm_1_log', 'fit_lm_2_log',
        'fit_lm_3_log', 'fit_lm_4_log']
print('Наименьшая ошибка на тестовой с k-fold10 у модели',
      fits[scores_log.index(max(scores_log))],
      ':\nMSE_kf10 =', np.around(-max(scores_log), 3))

```

Наименьшая ошибка на тестовой с k-fold10 у модели fit_lm_0_log :
MSE_kf10 = 0.198

Можно убедиться, что оценка MSE методом перекрёстной проверки по 10 блокам даёт результаты, практически идентичные методу LOOCV. При этом скорость у первого метода при 1204 наблюдениях выше на два порядка.

Самой точной среди моделей для `charges` оказалась `fit_lm_3`, а среди моделей для `charges_log` – `fit_lm_0_log`. Оценим точность прогноза по этим моделям на отложенные наблюдения.

In [50]:

```

#
# прогноз по fit_lm_3
# модель на всех обучающих наблюдениях
X = df3.drop(['charges'], axis=1)
y = df3.charges.values.reshape(-1, 1)

```

```

fit_lm_3 = lm.fit(X, y)

# значения y на отложенных наблюдениях
y = DF_predict[['charges']].values.reshape(-1, 1)
# матрица объясняющих на отложенных наблюдениях
X = DF_predict[['smokerYes', 'age', 'bmi']]
X.loc[:, 'bmi_smokerYes'] = X.loc[:, 'bmi'] * X.loc[:, 'smokerYes']
X = X.drop(['bmi'], axis=1)
# прогнозы
y_pred = fit_lm_3.predict(X)

# ошибка
MSE = sum((y - y_pred.reshape(-1, 1))**2) / len(y)
print('MSE модели fit_lm_3 на отложенных наблюдениях = %.2f' % MSE)

```

MSE модели fit_lm_3 на отложенных наблюдениях = 27756871.38

C:\Users\user\anaconda3\lib\site-packages\pandas\core\indexing.py:1597: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

self.obj[key] = value

C:\Users\user\anaconda3\lib\site-packages\pandas\core\indexing.py:1676: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

self._setitem_single_column(ilocs[0], value, pi)

In [51]:

```

# прогноз по fit_lm_log_0
# модель
X = df0.drop(['charges'], axis=1)
y = np.log(df0.charges).values.reshape(-1, 1)
fit_lm_0_log = lm.fit(X, y)

# значения y на отложенных наблюдениях
y = np.log(DF_predict[['charges']].values.reshape(-1, 1))
# матрица объясняющих на отложенных наблюдениях
X = DF_predict.drop(['charges', 'region'], axis=1)

# прогнозы
y_pred = fit_lm_0_log.predict(X)

# ошибка
MSE_log = sum((np.exp(y) - np.exp(y_pred).reshape(-1, 1))**2) / len(y)
print('MSE модели fit_lm_0_log на отложенных наблюдениях = %.2f' % MSE_log)

```

MSE модели fit_lm_0_log на отложенных наблюдениях = 79934529.87

Очевидно, на выборке для прогноза точнее модель `fit_lm_3`:

$$\hat{charges} = \hat{\beta}_0 + \hat{\beta}_1 \cdot smokerYes + \hat{\beta}_2 \cdot bmi \cdot smokerYes + \hat{\beta}_3 \cdot age$$

In [52]:

```

# ещё раз строим на обучающей
X = df3.drop(['charges'], axis=1)
y = df3.charges.values.reshape(-1, 1)
fit_lm_3 = lm.fit(X, y)

```

```
print('модель fit_lm_3:\n',
      'константа ', np.around(fit_lm_3.intercept_, 3),
      '\n объясняющие ', list(df3.drop(['charges'], axis=1).columns.values),
      '\n коэффициенты ', np.around(fit_lm_3.coef_, 3))
```

```
модель fit_lm_3:
константа [-2214.448]
объясняющие ['smokerYes', 'age', 'bmi_smokerYes']
коэффициенты [[-20774.593    269.6    1450.141]]
```

Прогноз на отложенные наблюдения по лучшей модели

Сделаем прогноз стоимости медицинской страховки по модели `fit_lm_3` на отложенные наблюдения.

```
In [53]: # ещё раз строим на обучающей
X = df3.drop(['charges'], axis=1)
y = df3.charges.values.reshape(-1, 1)
fit_lm_3 = lm.fit(X, y)
```

```
In [54]: # делаем данные для модели из отложенных наблюдений
df3 = DF_predict[['charges', 'smokerYes', 'age', 'bmi']]
df3.loc[:, 'bmi_smokerYes'] = df3.loc[:, 'bmi'] * df3.loc[:, 'smokerYes']
df3 = df3.drop(['bmi'], axis=1)
X = df3.drop(['charges'], axis=1)
y = df3.charges.values.reshape(-1, 1)
# прогноз
y_pred = fit_lm_3.predict(X)
# ошибка
MSE = sum((y - y_pred.reshape(-1, 1))**2) / len(y)
np.around(MSE, 2)
```

C:\Users\user\anaconda3\lib\site-packages\pandas\core\indexing.py:1597: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

self.obj[key] = value

C:\Users\user\anaconda3\lib\site-packages\pandas\core\indexing.py:1676: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

self._setitem_single_column(ilocs[0], value, pi)

```
Out[54]: array([27756871.38])
```

```
In [55]: # сравниваем ошибку со средним значением Y на отложенных
print('Ошибка модели на отложенных наблюдениях составила ',
      np.around(np.sqrt(MSE) / np.mean(y) * 100, 1)[0],
```

```
'% от среднего значения Y',  
sep='')
```

Ошибка модели на отложенных наблюдениях составила 37.2% от среднего значения Y