Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Отчет**

**по лабораторной работе № 6**

**по курсу «Технологии машинного обучения»**

**«Ансамбли моделей машинного обучения.»**

**ИСПОЛНИТЕЛЬ:**

Мехмандаров Мурад

Группа ИУ5-63

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "\_\_"\_\_\_\_\_\_\_\_\_\_\_2020г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "\_\_"\_\_\_\_\_\_\_\_\_\_\_2020г.

Москва 2020

**Цель лабораторной работы**

Изучение ансамблей моделей машинного обучения

# Задание

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train\_test\_split разделите выборку на обучающую и тестовую.
4. Обучите две ансамблевые модели. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.

# Ход выполнения лабораторной работы

Out[40]:

**import**

pandas

**as**

pd

**import**

seaborn

**as**

sns

**import**

numpy

**as**

np

**import**

matplotlib

.

pyplot

**as**

plt

**from**

sklearn

.

preprocessing

**import**

MinMaxScaler

,

LabelEncoder

**from**

sklearn

.

model\_selection

**import**

train\_test\_split

**from**

sklearn

.

metrics

**import**

f1\_score

,

precision\_score

,

recall\_score

,

accuracy\_score

**from**

sklearn

.

ensemble

**import**

RandomForestClassifier

**from**

sklearn

.

ensemble

**import**

GradientBoostingClassifier

**%**

matplotlib

inline

*#*

*Устанавливаем*

*тип*

*графиков*

sns

.

set

(

style

**=**

"ticks"

)

*#*

*Для*

*лучшего*

*качествоа*

*графиков*

**from**

IPython

.

display

**import**

set\_matplotlib\_formats

set\_matplotlib\_formats

(

"retina"

)

*#*

*Устанавливаем*

*ширину*

*экрана*

*для*

*отчета*

pd

.

set\_option

(

"display.width"

,

70

)

*#*

*Загружаем*

*данные*

data

**=**

pd

.

read\_csv

(

'../data/hotel\_bookings.csv'

)

data

.

head

()

**hotel is\_canceled lead\_time arrival\_date\_year arrival\_date\_month arrival\_date\_week\_numbe**

1. ResortHotel 0 342 2015 July 27
2. ResortHotel 0 737 2015 July 27
3. ResortHotel 0 7 2015 July 27
4. ResortHotel 0 13 2015 July 27
5. Resort 0 14 2015 July 27

Hotel

1. rows × 32 columns

data

**=**

data

.

dropna

(

)

data

.

isnull

().

sum

()

Out[41]:

hotel 0 is\_canceled 0 lead\_time 0 arrival\_date\_year 0 arrival\_date\_month 0 arrival\_date\_week\_number 0 arrival\_date\_day\_of\_month 0 stays\_in\_weekend\_nights 0 stays\_in\_week\_nights 0 adults 0 children 0 babies 0 meal 0 country 0 market\_segment 0 distribution\_channel 0 is\_repeated\_guest 0 previous\_cancellations 0 previous\_bookings\_not\_canceled 0 reserved\_room\_type 0 assigned\_room\_type 0 booking\_changes 0 deposit\_type 0 agent 0 company 0 days\_in\_waiting\_list 0 customer\_type 0 adr 0 required\_car\_parking\_spaces 0 total\_of\_special\_requests 0 reservation\_status 0 reservation\_status\_date 0 dtype: int64

data

.

isna

().

sum

(

)

Out[42]:

hotel 0 is\_canceled 0 lead\_time 0 arrival\_date\_year 0 arrival\_date\_month 0 arrival\_date\_week\_number 0 arrival\_date\_day\_of\_month 0 stays\_in\_weekend\_nights 0 stays\_in\_week\_nights 0 adults 0 children 0 babies 0 meal 0 country 0 market\_segment 0 distribution\_channel 0 is\_repeated\_guest 0 previous\_cancellations 0 previous\_bookings\_not\_canceled 0 reserved\_room\_type 0 assigned\_room\_type 0 booking\_changes 0 deposit\_type 0 agent 0 company 0 days\_in\_waiting\_list 0 customer\_type 0 adr 0 required\_car\_parking\_spaces 0 total\_of\_special\_requests 0 reservation\_status 0 reservation\_status\_date 0 dtype: int64 In [43]:

data

.

shape

Out[43]:

(217, 32)

[44]:

data

.

columns

Out[44]:

Index(['hotel', 'is\_canceled', 'lead\_time', 'arrival\_date\_year',

'arrival\_date\_month', 'arrival\_date\_week\_number',

'arrival\_date\_day\_of\_month', 'stays\_in\_weekend\_nights',

'stays\_in\_week\_nights', 'adults', 'children', 'babies',

'meal', 'country', 'market\_segment', 'distribution\_channel',

'is\_repeated\_guest', 'previous\_cancellations',

'previous\_bookings\_not\_canceled', 'reserved\_room\_type',

'assigned\_room\_type', 'booking\_changes', 'deposit\_type',

'agent', 'company', 'days\_in\_waiting\_list', 'customer\_type',

'adr', 'required\_car\_parking\_spaces',

'total\_of\_special\_requests', 'reservation\_status',

'reservation\_status\_date'], dtype='object') In [45]:

data

.

dtypes

Out[45]:

hotel object is\_canceled int64 lead\_time int64 arrival\_date\_year int64 arrival\_date\_month object arrival\_date\_week\_number int64 arrival\_date\_day\_of\_month int64 stays\_in\_weekend\_nights int64 stays\_in\_week\_nights int64 adults int64 children float64 babies int64 meal object country object market\_segment object distribution\_channel object is\_repeated\_guest int64 previous\_cancellations int64 previous\_bookings\_not\_canceled int64 reserved\_room\_type object assigned\_room\_type object booking\_changes int64 deposit\_type object agent float64 company float64 days\_in\_waiting\_list int64 customer\_type object adr float64 required\_car\_parking\_spaces int64 total\_of\_special\_requests int64 reservation\_status object reservation\_status\_date object dtype: object

Но сначала сделаем, кодирование категориальных признаков.

д , д р р р

In [46]:

le1

**=**

LabelEncoder

()

data

[

'hotel'

]

**=**

le1

.

fit\_transform

(

data

[

'hotel'

])

;

In [47]:

le2 **=** LabelEncoder() data['arrival\_date\_month'] **=** le2.fit\_transform(data['arrival\_date\_month']); In [48]:

le3

**=**

LabelEncoder

()

data

[

'meal'

]

**=**

le3

.

fit\_transform

(

data

[

'meal'

])

;

In [49]:

le4

**=**

LabelEncoder

()

data

[

'country'

]

**=**

le4

.

fit\_transform

(

data

[

'country'

])

;

In [50]:

le5 **=** LabelEncoder() data['market\_segment'] **=** le5.fit\_transform(data['market\_segment']);

In [51]:

le6 **=** LabelEncoder() data['distribution\_channel'] **=** le6.fit\_transform(data['distribution\_channel']);

In [52]:

le7 **=** LabelEncoder() data['reserved\_room\_type'] **=** le7.fit\_transform(data['reserved\_room\_type']);

In [53]:

le8 **=** LabelEncoder() data['deposit\_type'] **=** le8.fit\_transform(data['deposit\_type']);

In [54]:

le9 **=** LabelEncoder() data['customer\_type'] **=** le9.fit\_transform(data['customer\_type']);

In [55]:

le9 **=** LabelEncoder() data['reservation\_status'] **=** le9.fit\_transform(data['reservation\_status']);

In [56]:

le10 **=** LabelEncoder() data['reservation\_status\_date'] **=** le10.fit\_transform(data['reservation\_status\_date'] Набор данных не содержит пропусков

[58]:

*# Убедимся, что целевой признак*

*# для задачи бинарной классификации содержит только 0 и 1* data['is\_canceled'].unique() Out[58]:

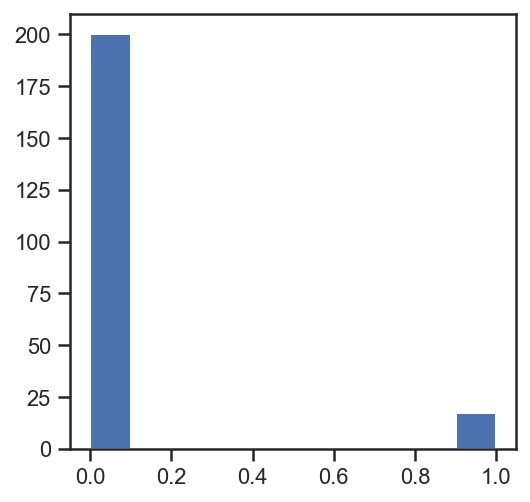
array([0, 1])

In

[59]:

In

[60]:



*#*

*Оценим*

*дисбаланс*

*классов*

*для*

*Occupancy*

fig

,

ax

**=**

plt

.

subplots

(

figsize

**=**

(

4

,

4

))

plt

.

hist

(

data

[

'is\_canceled'

])

plt

.

show

()

data

[

'is\_canceled'

].

value\_counts

()

Out[60]:

1. 200
2. 17

Name: is\_canceled, dtype: int64

In [61]:

*# посчитаем дисбаланс классов* total **=** data.shape[0]

class\_1, class\_0 **=** data['is\_canceled'].value\_counts() print('Класс 0 составляет {}%, а класс 1 составляет {}%.'

.format(round(class\_0 **/** total, 2)**\***100, round(class\_1 **/** total, 2)**\***100)) Класс 0 составляет 8.0%, а класс 1 составляет 92.0%.

Дисбаланса классов практически нет

## Проведем масштабирование данных

[85]:

ous\_bookings\_not\_canceled'

,

'required\_car\_parking\_spaces'

,

'country'

,

'meal'

,

'market

In [86]:

sc1

**=**

MinMaxScaler

(

)

sc1\_data

**=**

sc1

.

fit\_transform

(

data

[

scale\_cols

])

In [87]:

*# Добавим масштабированные данные в набор данных* **for** i **in** range(len(scale\_cols)): col **=** scale\_cols[i] new\_col\_name **=** col **+** '\_scaled' data[new\_col\_name] **=** sc1\_data[:,i] In [88]:

data

.

head

(

)

Out[88]:

**hotel is\_canceled lead\_time arrival\_date\_year arrival\_date\_month arrival\_date\_week\_numb**

**2392** 1 0 6 2015 10 4

**2697** 1 0 24 2015 10 4

**2867** 1 0 24 2015 9 4 **2877** 1 0 24 2015 9 4

**2878** 1 0 24 2015 9 4

5 rows × 50 columns

[89]:

corr\_cols\_1 **=** scale\_cols **+** ['is\_canceled'] corr\_cols\_1 Out[89]:

['lead\_time',

'arrival\_date\_year',

'arrival\_date\_week\_number',

'arrival\_date\_day\_of\_month',

'stays\_in\_weekend\_nights',

'stays\_in\_week\_nights',

'adults',

'children',

'babies',

'is\_repeated\_guest',

'previous\_cancellations',

'previous\_bookings\_not\_canceled', 'required\_car\_parking\_spaces',

'country',

'meal',

'market\_segment',

'reservation\_status',

'hotel',

'is\_canceled']

In [90]:

scale\_cols\_postfix **=** [x**+**'\_scaled' **for** x **in** scale\_cols] corr\_cols\_2 **=** scale\_cols\_postfix **+** ['is\_canceled'] corr\_cols\_2 Out[90]:

['lead\_time\_scaled',

'arrival\_date\_year\_scaled',

'arrival\_date\_week\_number\_scaled',

'arrival\_date\_day\_of\_month\_scaled',

'stays\_in\_weekend\_nights\_scaled',

'stays\_in\_week\_nights\_scaled',

'adults\_scaled',

'children\_scaled',

'babies\_scaled',

'is\_repeated\_guest\_scaled',

'previous\_cancellations\_scaled',

'previous\_bookings\_not\_canceled\_scaled', 'required\_car\_parking\_spaces\_scaled',

'country\_scaled',

'meal\_scaled',

'market\_segment\_scaled',

'reservation\_status\_scaled',

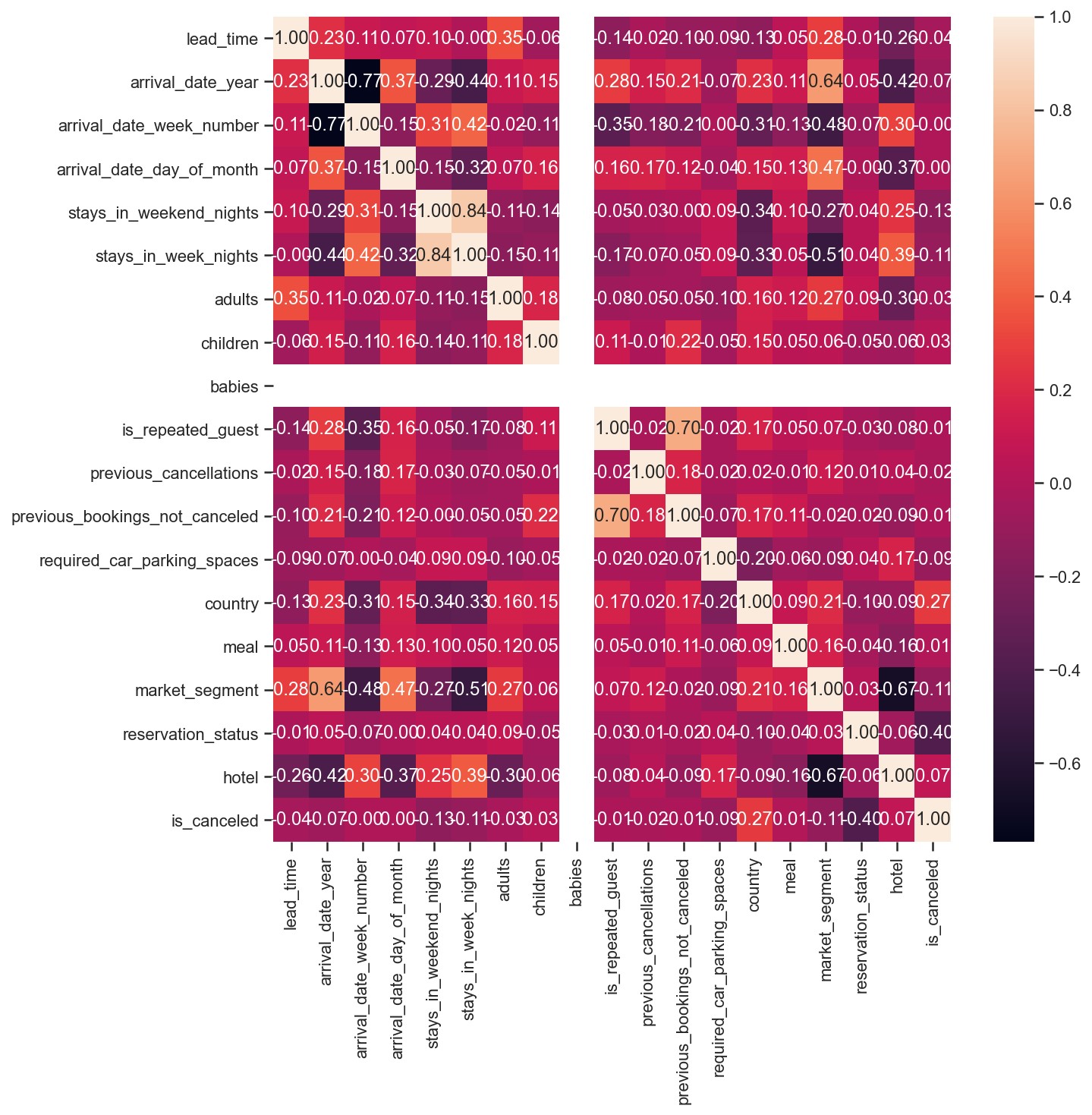
'hotel\_scaled',

'is\_canceled']

fig, ax **=** plt.subplots(figsize**=**(10,10)) sns.heatmap(data[corr\_cols\_1].corr(), annot**=True**, fmt**=**'.2f')

Out[91]:

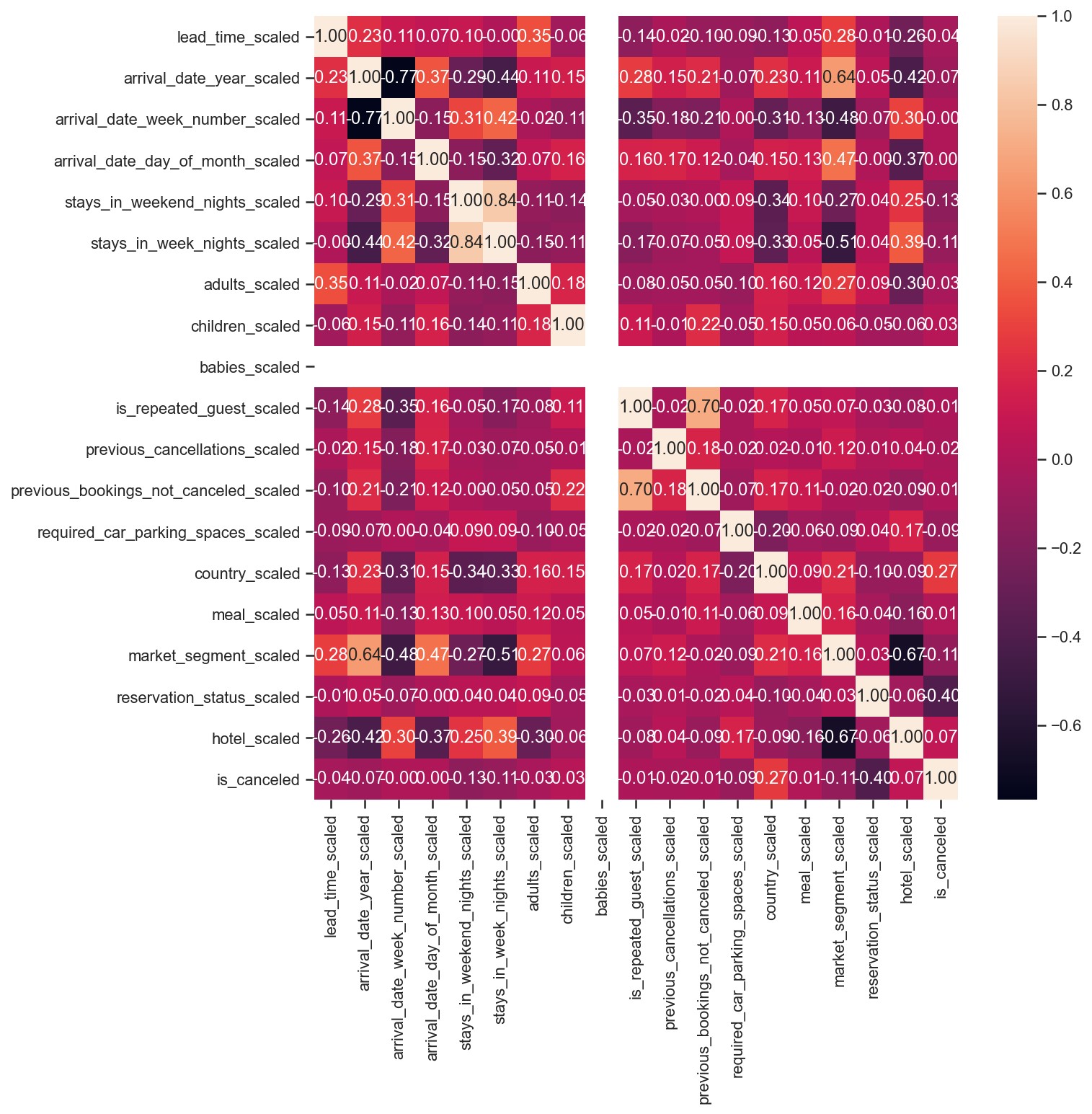
<matplotlib.axes.\_subplots.AxesSubplot at 0x125f4c510>



fig, ax **=** plt.subplots(figsize**=**(10,10)) sns.heatmap(data[corr\_cols\_2].corr(), annot**=True**, fmt**=**'.2f')

Out[92]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x12763f1d0>



На основе корреляционной матрицы можно сделать следующие выводы:

* Корреляционные матрицы для исходных и масштабированных данных совпадают;
* Оставим только 'country', 'meal', 'reservation\_status', 'hotel', так как они имеют корреляцию с целевым признаком.

*#*

*Числовые*

*колонки*

*для*

*масштабирования*

class\_cols

**=**

[

'country'

,

'meal'

,

'reservation\_status'

,

'hotel'

]

In

[94]:

X

**=**

data

[

class\_cols

]

y

**=**

data

[

'is\_canceled'

]

print

(

X

,

"\n"

)

print

(

y

)

country meal reservation\_status hotel 2392 15 0 1 1

2697 1 0 1 1

2867 8 0 1 1

1. 15 0 1 1
2. 12 0 1 1 ... ... ... ... ...

112499 5 0 1 0

113046 12 0 1 0

113082 12 0 1 0

113627 10 0 1 0

116451 15 0 1 0

[217 rows x 4 columns]

2392 0

2697 0

2867 0

1. 0
2. 0 ..

112499 0

113046 0

113082 0

113627 0

116451 0

Name: is\_canceled, Length: 217, dtype: int64

## Разделим выборку на обучающую и тестовую

In [95]:

*# С использованием метода train\_test\_split разделим выборку на обучающую и тестовую*

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.25, random\_sta print("X\_train:", X\_train.shape) print("X\_test:", X\_test.shape) print("y\_train:", y\_train.shape) print("y\_test:", y\_test.shape)

X\_train: (162, 4) X\_test: (55, 4) y\_train: (162,) y\_test: (55,) [96]:

**class** MetricLogger:

**def** \_\_init\_\_(self):

self.df **=** pd.DataFrame(

{'metric': pd.Series([], dtype**=**'str'),

'alg': pd.Series([], dtype**=**'str'), 'value': pd.Series([], dtype**=**'float')})

**def** add(self, metric, alg, value):

"""

Добавление значения

"""

*# Удаление значения если оно уже было ранее добавлено*

self.df.drop(self.df[(self.df['metric']**==**metric)**&**(self.df['alg']**==**alg)].inde

*# Добавление нового значения*

temp **=** [{'metric':metric, 'alg':alg, 'value':value}] self.df **=** self.df.append(temp, ignore\_index**=True**)

**def** get\_data\_for\_metric(self, metric, ascending**=True**): """

Формирование данных с фильтром по метрике

"""

temp\_data **=** self.df[self.df['metric']**==**metric]

temp\_data\_2 **=** temp\_data.sort\_values(by**=**'value', ascending**=**ascending) **return** temp\_data\_2['alg'].values, temp\_data\_2['value'].values

**def** plot(self, str\_header, metric, ascending**=True**, figsize**=**(5, 5)): """

Вывод графика

"""

array\_labels, array\_metric **=** self.get\_data\_for\_metric(metric, ascending) fig, ax1 **=** plt.subplots(figsize**=**figsize) pos **=** np.arange(len(array\_metric)) rects **=** ax1.barh(pos, array\_metric, align**=**'center', height**=**0.5, tick\_label**=**array\_labels) ax1.set\_title(str\_header) **for** a,b **in** zip(pos, array\_metric): plt.text(0.5, a**-**0.05, str(round(b,3)), color**=**'white') plt.show()

In [97]:

*#*

*Сохранение*

*метрик*

metricLogger

**=**

MetricLogger

(

)

[98]:

**def** test\_model(model\_name, model, metricLogger): model.fit(X\_train, y\_train) y\_pred **=** model.predict(X\_test)

accuracy **=** accuracy\_score(y\_test, y\_pred) f1 **=** f1\_score(y\_test, y\_pred) precision **=** precision\_score(y\_test, y\_pred) recall **=** recall\_score(y\_test, y\_pred)

metricLogger.add('precision', model\_name, precision) metricLogger.add('recall', model\_name, recall) metricLogger.add('f1', model\_name, f1)

metricLogger.add('accuracy', model\_name, accuracy)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*') print(model) print(model\_name) print("accuracy:", accuracy) print("f1\_score:", f1) print("precision\_score:", precision) print("recall:", recall) print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

## Обучим модели

Будем использовать модели: случайный лес, градиентный бустинг

[99]:

test\_model('Random forest', RandomForestClassifier(), metricLogger) test\_model('GB', GradientBoostingClassifier(), metricLogger)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=Non e,

criterion='gini', max\_depth=None, max\_features

='auto', max\_leaf\_nodes=None, max\_samples=None, min\_impurity\_decrease=0.0, min\_impurity\_split=N

one,

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=100, n\_jobs=None, oob\_score=False, random\_state=Non

e,

verbose=0, warm\_start=False) Random forest accuracy: 1.0 f1\_score: 1.0 precision\_score: 1.0 recall: 1.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', in it=None,

learning\_rate=0.1, loss='deviance', max\_dep

th=3,

max\_features=None, max\_leaf\_nodes=None, min\_impurity\_decrease=0.0, min\_impurity\_spl

it=None,

min\_samples\_leaf=1, min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0, n\_estimators=

100, n\_iter\_no\_change=None, presort='deprecate

d', random\_state=None, subsample=1.0, tol=0.000

1, validation\_fraction=0.1, verbose=0, warm\_start=False) GB accuracy: 1.0 f1\_score: 1.0 precision\_score: 1.0 recall: 1.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* In [100]:

*#*

*Метрики*

*качества*

*модели*

metrics

**=**

metricLogger

.

df

[

'metric'

].

unique

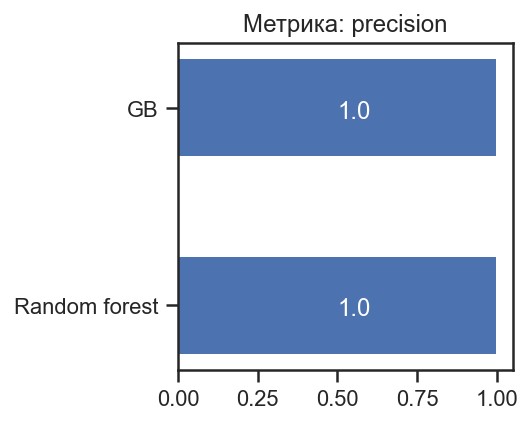
()

metrics

Out[100]:

array(['precision', 'recall', 'f1', 'accuracy'], dtype=object)

[101]:



*#*

*Построим*

*графики*

*метрик*

*качества*

*модели*

**for**

metric

**in**

metrics

:

metricLogger

.

plot

(

'

Метрика

:

'

**+**

metric

,

metric

,

figsize

**=**

(

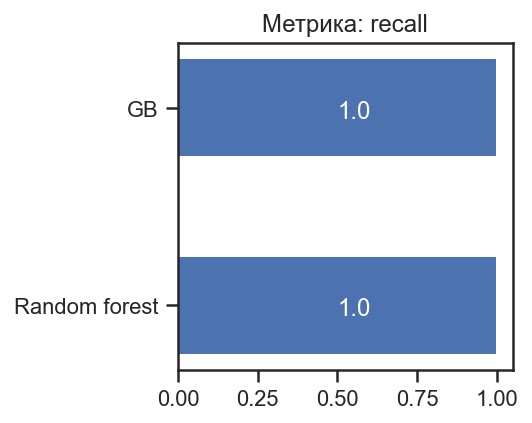
3

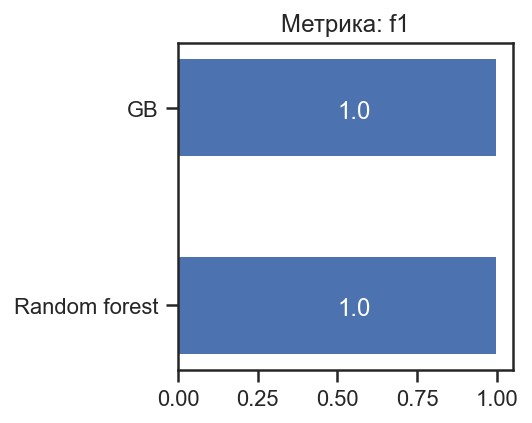
,

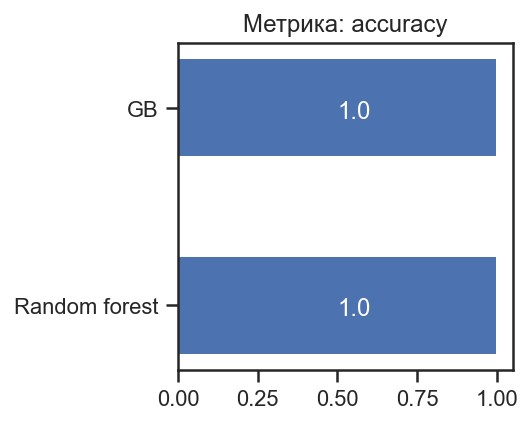
3

)

)







**#### Вывод: на основании четырех метрик из четырех используемых, модели оказались**

**одинаково эффективно.**