Рекомендация тарифов

```
Оператор мобильной связи «Мегалайн» выяснил: многие клиенты пользуются архивными тарифами. Они хотят построить систему, с
пособную проанализировать поведение клиентов и предложить пользователям новый тариф: «Смарт» или «Ультра».
В нашем распоряжении данные о поведении клиентов, которые уже перешли на эти тарифы (из проекта курса «Статистический ана
лиз данных»). Нужно построить модель для задачи классификации, которая выберет подходящий тариф.

Описание данных

Каждый объект в наборе данных — это информация о поведении одного пользователя за месяц. Известно:
```

calls — количество звонков,

minutes — суммарная длительность звонков в минутах,

messages — количество sms-сообщений,

mb_used — израсходованный интернет-трафик в Мб,

is_ultra — каким тарифом пользовался в течение месяца («Ультра» — 1, «Смарт» — 0).

```
Построим модель с максимально большим значением *accuracy*(не менее 0.75). Проверим *accuracy* на тестовой выборке.
```

1. Откроем и изучим файл

In [77]:
```python
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split #импортируем функцию train_test_split из библиотеки sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
```

In [78]:
```python
df=pd.read_csv("/datasets/users_behavior.csv") #прочитаем файл
df.info() # изучим общую информацию по дата фрейму
df.head(100)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3214 entries, 0 to 3213
Data columns (total 5 columns):
calls       3214 non-null float64
minutes     3214 non-null float64
messages    3214 non-null float64
mb_used     3214 non-null float64
is_ultra    3214 non-null int64
dtypes: float64(4), int64(1)
memory usage: 125.7 KB
```

Out[78]:

	calls	minutes	messages	mb_used	is_ultra
0	40.0	311.90	83.0	19915.42	0
1	85.0	516.75	56.0	22696.96	0
2	77.0	467.66	86.0	21060.45	0
3	106.0	745.53	81.0	8437.39	1
4	66.0	418.74	1.0	14502.75	0
...	...	...	...	...	...
95	101.0	666.90	57.0	16777.76	1
96	29.0	267.55	29.0	16996.83	0
97	83.0	538.83	60.0	13721.94	0
98	67.0	454.43	31.0	19776.50	0
99	66.0	434.59	17.0	9613.39	0

100 rows × 5 columns

Вывод:

Данные прошли предобработку. Добалили необходимые библиотеки, изучили общую информацию по датафрейму.

2. Разобьем данные на выборки

In [79]:
```python
#Разделим исходные данные на обучающую, валидационную и тестовую выборки.
df_train, df_valid_40 = train_test_split(df, test_size=0.4, random_state=12345)
df_valid, df_test = train_test_split(df_valid_40, test_size=0.5, random_state=12345)
#изучим выборки
df_train.info()
df_valid.info()
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1928 entries, 3027 to 482
Data columns (total 5 columns):
calls       1928 non-null float64
minutes     1928 non-null float64
messages    1928 non-null float64
mb_used     1928 non-null float64
is_ultra    1928 non-null int64
dtypes: float64(4), int64(1)
memory usage: 90.4 KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 643 entries, 1386 to 3197
Data columns (total 5 columns):
calls       643 non-null float64
minutes     643 non-null float64
messages    643 non-null float64
mb_used     643 non-null float64
is_ultra    643 non-null int64
dtypes: float64(4), int64(1)
memory usage: 30.1 KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 643 entries, 160 to 2313
Data columns (total 5 columns):
calls       643 non-null float64
minutes     643 non-null float64
messages    643 non-null float64
mb_used     643 non-null float64
is_ultra    643 non-null int64
dtypes: float64(4), int64(1)
memory usage: 30.1 KB
```

In [80]:
```python
#создадим переменные для признаков и целевого признака для 3х выборок:
#для обучающей
features_train = df_train.drop(['is_ultra'], axis=1)
target_train = df_train['is_ultra']
#для валидационной
features_valid = df_valid.drop(['is_ultra'], axis=1)
target_valid = df_valid['is_ultra']
#для тестовой
features_test=df_test.drop(['is_ultra'], axis=1)
target_test=df_test['is_ultra']
```

Вывод:

Разделили исходные данные на обучающую, валидационную и тестовую выборки методом train_test_split в 2 подхода в пропорциях 3:1:1. Проверили методом info() правильно ли разделена выборка. Создали переменные для признаков и целевого признака для 3х выборок.

3. Исследуем модели

In [81]:
```python
#Исследуем качество разных моделей, меняя гиперпараметры.

#для модели решающего дерева
#в цикле проверим гиперпараметр глубины дерева от 1 до 30, кратно 2
depth_and_score=pd.DataFrame(columns=["depth", 'accuracy_valid', 'accuracy_train'])
i=0
for depth in range(1,31,2):
    model_DTC = DecisionTreeClassifier(random_state=12345, max_depth=depth)
    model_DTC.fit(features_train, target_train) #обучим модель
    predictions_DTC_valid=model_DTC.predict(features_valid) #предскажем по валидационной выборке
    predictions_DTC_train=model_DTC.predict(features_train) #предскажем по тестовой выборке
    accuracy_valid_DTC= accuracy_score(target_valid, predictions_DTC_valid)
    accuracy_train_DTC=accuracy_score(target_train, predictions_DTC_train)
    depth_and_score.loc[i]=[depth, accuracy_valid_DTC, accuracy_train_DTC]  #построим датафрейм с данными по accuracy на валидац
ионной и тестовой выборке
    i+=1
print(depth_and_score)
```

```
    depth  accuracy_valid  accuracy_train
0     1.0        0.754277        0.757780
1     3.0        0.785381        0.807573
2     5.0        0.779160        0.820021
3     7.0        0.782271        0.855809
4     9.0        0.782271        0.881224
5    11.0        0.762053        0.906639
6    13.0        0.755832        0.941909
7    15.0        0.746501        0.967842
8    17.0        0.735614        0.984440
9    19.0        0.727838        0.989108
10   21.0        0.727838        0.995332
11   23.0        0.716952        0.998963
12   25.0        0.713841        1.000000
13   27.0        0.713841        1.000000
14   29.0        0.713841        1.000000
```

Здесь мы наглядно можем проследить, как с увеличением глубины модель переобучается.

In [82]:
```python
DTC_3_accuracy_valid=depth_and_score['accuracy_valid'][3]
#Запишем в переменные значения, чтобы потом сделать таблицу сравнительную
DTC_3_accuracy_test=depth_and_score['accuracy_train'][3]
```

In [83]:
```python
#для модели случайного леса
#в цикле проверим качество модели для гиперпараметра количество оценщиков от 1 до 100, кратно 10
estim_and_score=pd.DataFrame(columns=["estimators","max_depth",'accuracy_valid','accuracy_train'])
i=0
for estim in range(1,101,10):
    for depth_RFC in range (1,10,2):
        model_RFC = RandomForestClassifier(random_state=12345, n_estimators=estim, max_depth=depth_RFC)
        model_RFC.fit(features_train, target_train) #обучим модель
        predictions_RFC_valid=model_RFC.predict(features_valid)
        predictions_RFC_train=model_RFC.predict(features_train)
        accuracy_valid_RFC= accuracy_score(target_valid, predictions_RFC_valid) # расчёт точности на валидационной выборке
        accuracy_train_RFC=accuracy_score(target_train, predictions_RFC_train)
        estim_and_score.loc[i]=[estim, depth_RFC, accuracy_valid_RFC, accuracy_train_RFC]
        i+=1
print(estim_and_score)
```

```
    estimators  max_depth  accuracy_valid  accuracy_train
0          1.0        1.0        0.754277        0.741701
1          1.0        3.0        0.785381        0.787344
2          1.0        5.0        0.776050        0.807054
3          1.0        7.0        0.777605        0.828838
4          1.0        9.0        0.779160        0.845436
5         11.0        1.0        0.754277        0.744813
6         11.0        3.0        0.783826        0.810166
7         11.0        5.0        0.785381        0.824689
8         11.0        7.0        0.796267        0.853734
9         11.0        9.0        0.785381        0.875519
10        21.0        1.0        0.763608        0.751556
11        21.0        3.0        0.786936        0.810166
12        21.0        5.0        0.788491        0.827282
13        21.0        7.0        0.800933        0.858402
14        21.0        9.0        0.793157        0.884336
15        31.0        1.0        0.788491        0.781120
16        31.0        3.0        0.786936        0.808610
17        31.0        5.0        0.793157        0.824170
18        31.0        7.0        0.802488        0.859440
19        31.0        9.0        0.790047        0.883299
20        41.0        1.0        0.776050        0.767116
21        41.0        3.0        0.786936        0.808610
22        41.0        5.0        0.793157        0.825726
23        41.0        7.0        0.802488        0.863071
24        41.0        9.0        0.793157        0.884336
25        51.0        1.0        0.762053        0.764004
26        51.0        3.0        0.788491        0.808091
27        51.0        5.0        0.793157        0.825726
28        51.0        7.0        0.802488        0.863071
29        51.0        9.0        0.794712        0.885373
30        61.0        1.0        0.763608        0.764523
31        61.0        3.0        0.786936        0.808091
32        61.0        5.0        0.791602        0.823651
33        61.0        7.0        0.799378        0.864108
34        61.0        9.0        0.796267        0.885373
35        71.0        1.0        0.762053        0.764004
36        71.0        3.0        0.788491        0.807573
37        71.0        5.0        0.793157        0.824689
38        71.0        7.0        0.802488        0.863071
39        71.0        9.0        0.796267        0.883817
40        81.0        1.0        0.763608        0.764523
41        81.0        3.0        0.788491        0.808091
42        81.0        5.0        0.793157        0.824689
43        81.0        7.0        0.799378        0.861515
44        81.0        9.0        0.796267        0.885373
45        91.0        1.0        0.766719        0.761929
46        91.0        3.0        0.788491        0.808091
47        91.0        5.0        0.793157        0.825207
48        91.0        7.0        0.800933        0.863071
49        91.0        9.0        0.794712        0.884336
```

In [84]:
```python
best_RFC=estim_and_score.query('accuracy_valid==accuracy_valid.max()')
best_RFC
```

Out[84]:

	estimators	max_depth	accuracy_valid	accuracy_train
18	31.0	7.0	0.802488	0.859440
23	41.0	7.0	0.802488	0.863071
28	51.0	7.0	0.802488	0.863071
38	71.0	7.0	0.802488	0.863071

In [85]:
```python
RFC_31_accuracy_valid=best_RFC['accuracy_valid'][18]
RFC_31_accuracy_train=best_RFC['accuracy_train'][18]
```

In [86]:
```python
#для модели логистической регрессии
model_LR = LogisticRegression(random_state=12345)
model_LR.fit(features_train, target_train) #обучим модель
predictions_LR_valid=model_LR.predict(features_valid)
accuracy_valid_LR= accuracy_score(target_valid, predictions_LR_valid) #код расчёта на валидационной выборке
print(accuracy_valid_LR)
```

```
0.7589424572317263
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to
'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Качество моделей будем исследовать при помощи метрики accuracy - Отношение числа правильных ответов к размеру тестовой выборки или «доля правильных ответов». Для модели решающего дерева было исследовано значение качества модели для гиперпараметра глубина дерева от 1 до 30, кратная 2. Максимальная доля правильных ответов для валидационной выборки наблюдается у максимальной глубины 3 - 78,53%. Для модели случайного леса было исследовано значение качества модели для гиперпараметра n_estimators - количество оценщиков, от 1 до 100, кратные 10. Максимальная доля правильных ответов для валидационной выборки наблюдается при n_estimators= 21 - 79,31%. Для модели логистической регрессии значение максимальной доли правильных ответов 75,89%.

4. Проверим модель на тестовой выборке

In [87]:
```python
#для модели решающего дерева с гиперпараметром глубины дерева 3
model_DTC_3 = DecisionTreeClassifier(random_state=12345, max_depth=3)
model_DTC_3.fit(features_train, target_train) #обучим модель
predictions_DTC_test_3=model_DTC_3.predict(features_test)
accuracy_test_DTC_3=accuracy_score(target_test, predictions_DTC_test_3)
print("Доля правильных ответов для модели решающего дерева с max_depth=3:", accuracy_test_DTC_3)
```

```
Доля правильных ответов для модели решающего дерева с max_depth=3: 0.7791601866251944
```

In [88]:
```python
#для модели случайного леса с гиперпараметром количества оценщиков 31
model_RFC_31 = RandomForestClassifier(random_state=12345, n_estimators=31, max_depth=7)
model_RFC_31.fit(features_train, target_train) #обучим модель
predictions_RFC_test_31=model_RFC_31.predict(features_test)
accuracy_test_RFC_31=accuracy_score(target_test, predictions_RFC_test_31)
print("Доля правильных ответов для модели случайного леса с n_estimators=31, max_depth=7:", accuracy_test_RFC_31)
```

```
Доля правильных ответов для модели случайного леса с n_estimators=31, max_depth=7: 0.8040435458786936
```

In [89]:
```python
#для модели логистической регрессии
predictions_LR_test=model_LR.predict(features_test)
accuracy_test_LR=accuracy_score(target_test, predictions_LR_test)
print("Доля правильных ответов для модели логистической регрессии:", accuracy_test_LR)
```

```
Доля правильных ответов для модели логистической регрессии: 0.7402799377916018
```

In [90]:
```python
results_valid = {'DecisionTreeClassifier' : DTC_3_accuracy_valid,
                 'RandomForestClassifier' : RFC_31_accuracy_valid,
                 'LogisticRegression' : accuracy_valid_LR}
accuracy_valid=pd.Series(results_valid)
results_test = {'DecisionTreeClassifier' : DTC_3_accuracy_test,
                'RandomForestClassifier' : accuracy_test_RFC_31,
                'LogisticRegression' : accuracy_test_LR}
accuracy_test=pd.Series(results_test)
results =pd.DataFrame({'accuracy_valid': accuracy_valid, 'accuracy_test': accuracy_test})
results
```

Out[90]:

	accuracy_valid	accuracy_test
DecisionTreeClassifier	0.782271	0.855809
RandomForestClassifier	0.802488	0.804044
LogisticRegression	0.758942	0.740280

Вывод:

После проверки моделей на тестовой выборке можно сделать вывод о том, что модель случайного леса является самой точной на валидационной выборке 79%, хотя и проигрывает немного модели дерево решений по проверке на тестовой выборке 77%. У дерева решений высокая точность 78,5% и высокая скорость, в отличие от скорости модели случайного леса. Модель логистической регрессии не дотягивает до нашего условия точности 75%.

5. Проверим модели на адекватность

In [91]:
```python
dummy_clf=DummyClassifier(strategy='uniform')
dummy_clf.fit(features_train, target_train)
#проверим на валидационной выборке
dummy_predict_valid=dummy_clf.predict(features_valid)
accuracy_dummy_clf_valid=accuracy_score(target_valid, dummy_predict_valid)
print("Значение accuracy на валидационной выборке:", accuracy_dummy_clf_valid)
```

```
Значение accuracy на валидационной выборке: 0.5116640746500778
```

In [92]:
```python
#проверим на тестовой выборке
dummy_predict_test=dummy_clf.predict(features_test)
accuracy_dummy_clf_test=accuracy_score(target_test, dummy_predict_test)
accuracy_dummy_clf_test
print("Значение accuracy на тестовой выборке:", accuracy_dummy_clf_test)
```

```
Значение accuracy на тестовой выборке: 0.5085536547433903
```

Вывод:

Значения у нашей модели лучше, чем у случайной модели.

Вывод:

В данном проекте нам повезло и нам достались данные прошедшие предобработку, так что нам осталось только разделить данные на 3 выборки. Мы исследовали 3 модели на качество, меняя гиперпараметры: модель дерева решений, случайный лес и логистической регрессии. Качество моделей было исследованно при помощи метрики accuracy - Отношение числа правильных ответов к размеру тестовой выборки или «доля правильных ответов». Для модели решающего дерева было исследовано значение качества модели для гиперпараметра глубина дерева от 1 до 30, кратная 2. Максимальная доля правильных ответов для валидационной выборки наблюдается у максимальной глубины 3 - 78,53%. Для модели случайного леса было исследовано значение качества модели для гиперпараметра n_estimators - количество оценщиков, от 1 до 100, кратные 10. Максимальная доля правильных ответов для валидационной выборки наблюдается при n_estimators= 21 - 79,31%. Для модели логистической регрессии значение максимальной доли правильных ответов 75,89%. После проверки моделей на тестовой выборке можно сделать вывод о том, что модель случайного леса является самой точной на валидационной выборке 79%, хотя и проигрывает немного модели дерево решений по проверке на тестовой выборке 77%. У дерева решений высокая точность 78,5% и высокая скорость, в отличие от скорости модели случайного леса. Модель логистической регрессии не дотягивает до нашего условия точности 75%. Я бы все-таки остановилась на модели случайного леса. Также в 5 части проекта мы сравнили данные точности случайной модели с нашими тремя моделями. Точность наших моделей выше.