Данные и постановка задачи

Исходные данные:

- Информация об уникальных сессиях пользователях. Файл ga_sessions.csv размерностью 1860042 x 18
- Информация о действиях пользователей. Файл ga_hits.csv размерностью 15726470 x 11
- Информация о том, какие действия являются целевыми
- Сессия считается конверсионной, если хотя бы одно действие в ее рамках было целевым
- Перечень уникальных значений из колонок ‘utm_source’ и ‘utm_medium, которые указывают соответственно на рекламу в социальных сетях и органический трафик

Задачи:

- Создать несколько моделей машинного обучения, произведя предварительную обработку данных и генерацию новых признаков
- Достигнуть показателя метрики ROC_AUC > 0.65, в том числе с помощью тюнинга гиперпараметров
- Выбрать лучшую модель и подготовить ее для последующего внедрения

Объединение датафреймов

Алгоритм объединения:

- Добавим к датафрейму df_hits целевую колонку 'target'
- Положим в неё значение '1', если действие из колонки 'event_action' является целевым (из перечня действий, согласно условию), в противном случае - положим значение ‘0’
- Создадим вспомогательный датафрейм из двух колонок ‘session_id’ и ‘target’ и удалим из него строки-дубликаты (так как в рамках одной сессии могло быть несколько как целевых, так и нецелевых действий)
- Сделаем его группировку с агрегирующей функцией SUM, благодаря чему все значения ‘session_id’ станут уникальными, и им будет соответствовать либо ‘1’, либо ‘0’ из колонки ‘target’
- Объединим его с df_sessions по колонке ‘session_id’ методом INNER
- Имеем финальный датасет размерностью 1732266 x 19. Количество строк уменьшилось, так как объединение произошло по сессиям, которые имелись только в обеих таблицах

Анализ объединенного датафрейма

Классы целевой колонки очень разбалансированы - процент конвертации сессии в целевое действие около 2,9%.

```
df.target.value_counts()
```

```
0    1681952
1      50314
```

Именно поэтому в качестве метрики будущей модели правильным будет использование метрики ROC_AUC.

Ввиду того, что использование порядковых номеров ("айдишников") при обучении модели является неправильной практикой, а также для экономии ресурсов ПК при работе с большими данными, сразу же удалим колонки **‘session_id’ и ‘client_id’.**

Перечень колонок с количеством и процентом пропусков в них

```
df.isna().sum().sort_values(ascending=False)
```

device_model	1717204	99.13
utm_keyword	1020752	58.93
device_os	1013964	58.53
device_brand	347196	20.04
utm_adcontent	304137	17.56
utm_campaign	195287	11.27
utm_source	76	0.00
target	0	0.00
visit_time	0	0.00
visit_number	0	0.00
utm_medium	0	0.00
device_category	0	0.00
geo_city	0	0.00
device_screen_resolution	0	0.00
device_browser	0	0.00
geo_country	0	0.00
visit_date	0	0.00

Обработка пропусков

КОЛОНКА	**ДЕЙСТВИЕ НАД ПРОПУСКАМИ** (и неинформативными значениями ‘not set’)
device_model, utm_keyword	Полностью удаляем колонки, так процент пропусков очень большой, а заполнение чем-либо только навредит модели.
utm_source, utm_medium	Заполняем модой.
utm_compaign, utm_adcontent	Заполняем значением ‘other’.
device_brand	‘Apple’, если device_os ==’ Macintosh’ or ’iOS’ ‘Apple’, если device_browser == ’Safari’ ‘Samsung’, если device_os == ’Tizen’ ‘Samsung’, если device_browser == ’Sumsung Internet’ Оставшиеся пропуски заполняем значением ‘unknown’
device_os	‘iOS’, если device_brand == ’Apple’ ‘iOS’, если device_browser ==’ Safari’ ‘Windows’, если device_browser == ’Internet Explorer’ or ‘Edge’ Оставшиеся пропуски заполняем значением ‘Android’
geo_city	Значения ‘not set’ заполним наиболее часто встречающимся названием города определенной страны, если такого нет - то названием самой страны

Объединение мелких категорий в категориальных признаках

ЦЕЛЬ: Снизить размерность датасета, который образуется в будущем после “горячего кодирования” для экономии ресурсов пк, избавить модель от незначительного “шума”

АЛГОРИТМ ОБЪЕДИНЕНИЯ:

```
# Подсчет количества значений.
df.utm_source.value_counts()
```

```
# Создание списка категорий с количеством значений менее 150.
lst = df.utm_source.value_counts().loc[lambda x : x < 150].index.tolist()
# Создание новой колонки 'utm_source_upd' с присвоением категориям из списка lst значения 'rare'.
df['utm_source_upd'] = df.apply(lambda x: 'rare' if x.utm_source in lst else x.utm_source, axis=1)
```

```
# Удалим колонки, которые использовали для создания новых, и которые нам больше не пригодятся.
df = df.drop(columns=['utm_source', 'utm_medium', 'utm_campaign', 'utm_adcontent', 'device_brand', 'device_browser'])
df.head()
```

РЕЗУЛЬТАТ ОБЪЕДИНЕНИЯ:

column	quantity_limit	n_categories_before	n_categories_after
utm_source	150	280	57
utm_medium	120	54	25
utm_campaign	400	407	118
utm_adcontent	150	281	72
device_brand	70	200	51
device_browser	100	55	14

Генерация признаков

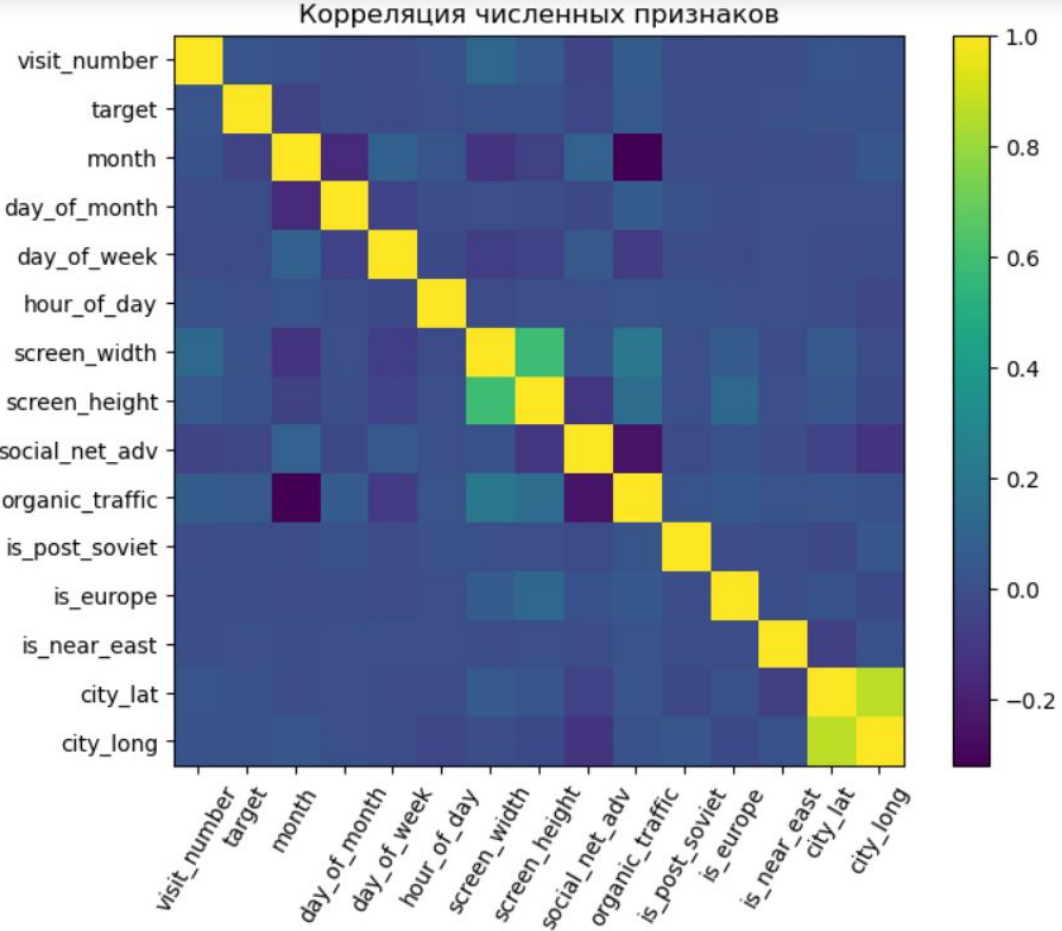
ПРИЗНАК	ОСНОВА	ФОРМУЛА
month	visit_date	Сперва приведем колонку 'visit_date' к типу 'datetime': df['visit_date'] = pd.to_datetime(df['visit_date']) Генерация признака: df['month'] = df['visit_date'].dt.month
day_of_month	visit_date	df['day_of_month'] = df['visit_date'].dt.day
day_of_week	visit_date	df['day_of_week'] = df['visit_date'].dt.dayofweek
hour_of_day	visit_time	df['hour_of_day'] = df.visit_time.apply(lambda x: x.split(':')[0]).astype(int)
screen_width	device_screen_resolution	df['screen_width'] = df.device_screen_resolution.apply(lambda x: x.split('x')[0]).astype(int) Строки с выбросами, где ‘screen_width’==0, удаляем
screen_height	device_screen_resolution	df['screen_height'] = df.device_screen_resolution.apply(lambda x: x.split('x')[1]).astype(int) Строки с выбросами, где ‘screen_height’ нереально большой, удаляем
screen_area	screen_width/height	df['screen_area'] = df['screen_width'] * df['screen_height']

Генерация признаков

ПРИЗНАК	ОСНОВА	ФОРМУЛА
device_age	screen_area	df.apply(lambda x: "new_top" if x.screen_area > 2000000 else ("new" if 800000 < x.screen_area <= 2000000 else ("average" if 300000 < x.screen_area <= 800000 else "old")), axis=1)
social_net_adv	utm_source_upd	df.apply(lambda x: 1 if x.utm_source_upd in lst else 0, axis=1) где lst - список с перечнем значений, относящихся к соцсетям
organic_traffic	utm_medium_upd	df.apply(lambda x: 1 if x.utm_medium_upd in lst else 0, axis=1) где lst = ['organic', 'referral', '(none)'], согласно начальным данным
is_post_soviet, is_europe, is_near_east	geo_country	df.apply(lambda x: 1 if x.geo_country in lst else 0, axis=1) где lst - список соответственно постсоветских, европейских, ближневосточных стран
city_lat, city_long	geo_city	С помощью библиотеки ‘geopy’ и api-сервиса ‘nominatim’ для каждого города извлечем долготу и широту и положим в соответствующие колонки

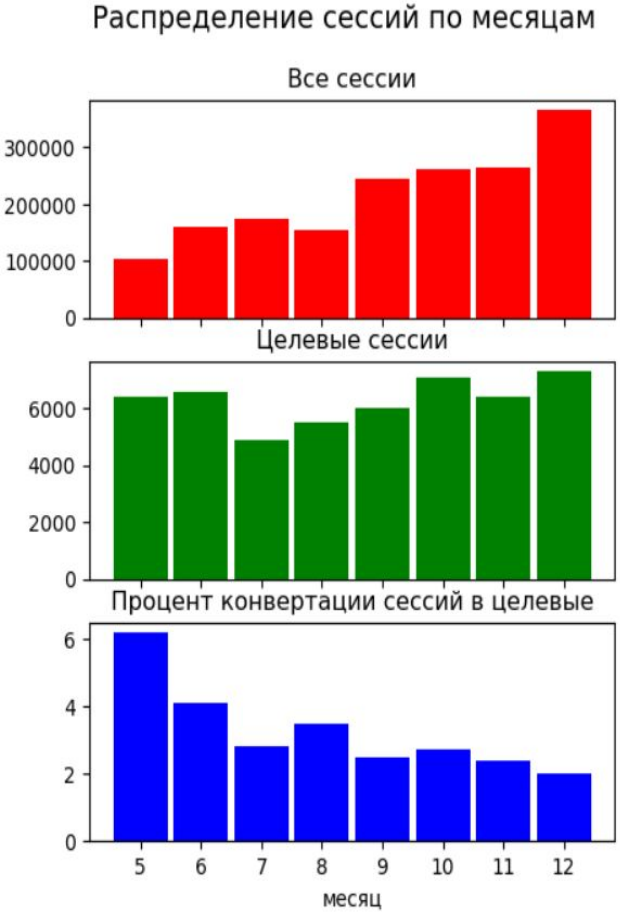
Удалим колонки, которые использовали для генерации новых признаков (visit_date, visit_time, screen_area, device_screen_resolution, geo_city, geo_city_upd, geo_info.

Корреляция признаков



- Целевая переменная ‘target’ не коррелирует ни с одним из признаков, что не окажет негативного влияния на будущее моделирование
- Есть некоторая корреляция между шириной/высотой экрана и географическими координатами, но для пространственных признаков это допустимо

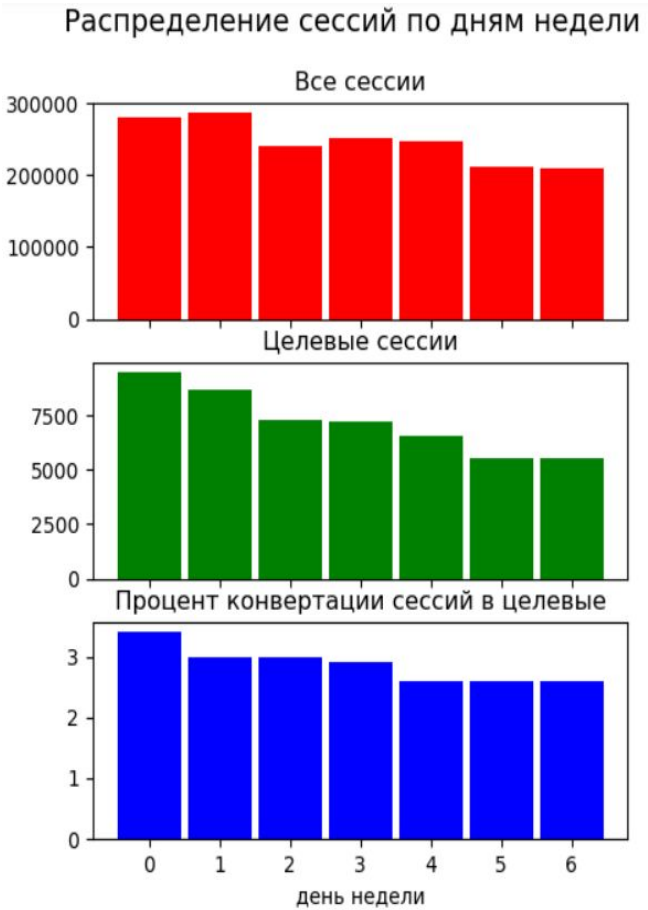
Анализ зависимостей



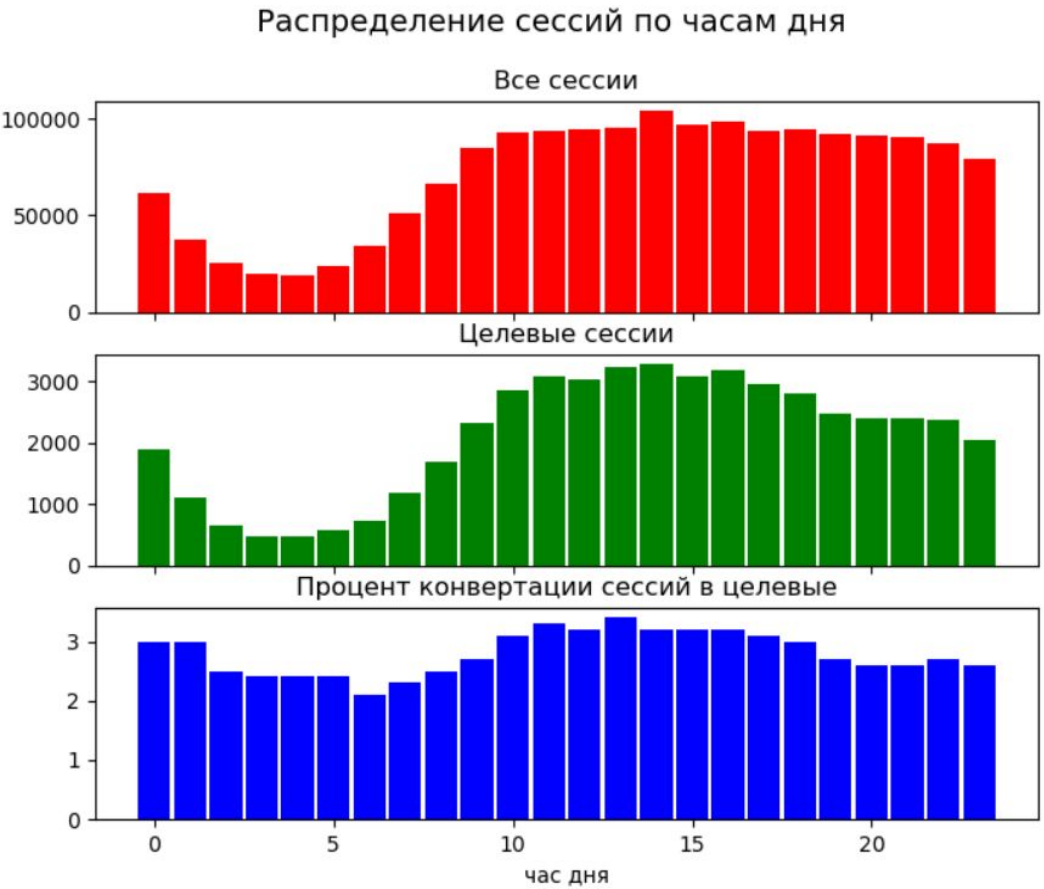
Общее число сессий росло почти каждый месяц, но процент конвертации по месяцам заметно снижался. Это может говорить о заинтересованности сервисом со стороны потенциальных клиентов, но невозможности найти подходящее предложение.

Количество посещений выше в начале недели и ниже в выходные дни, что логично.

Количество целевых сессий и их конвертация в еще большей степени выше в начале недели, чем в конце. Это говорит о предпочтении людей решать дела “с новыми силами”.



Анализ зависимостей



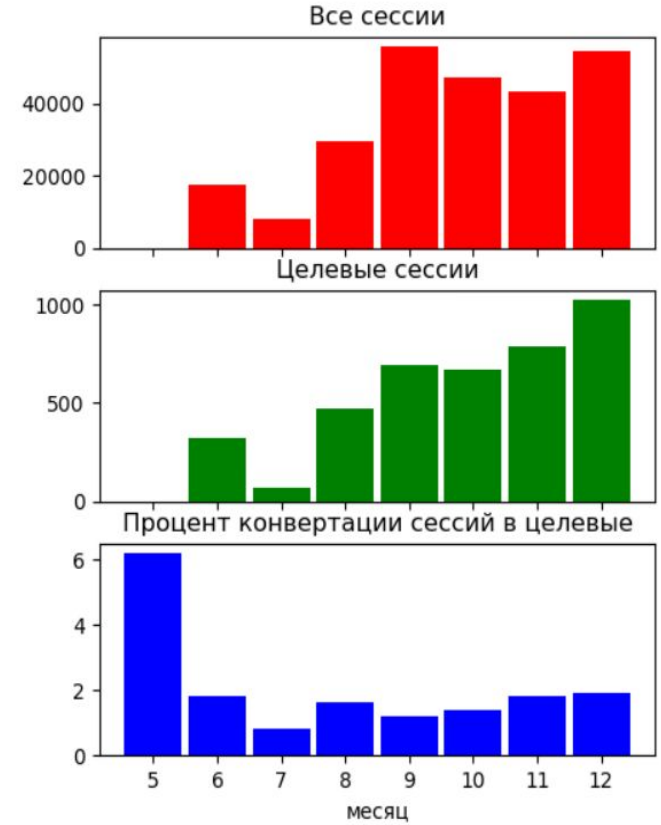
Очевидна зависимость как общего количества сессий, так и целевых, от времени суток. Это обусловлено физиологией человека.

Процент конвертации выше в часы бодрствования - клиенту удобнее решать ответственные вопросы в привычное время.

Также в некоторое время суток могут быть недоступны некоторые сервисы, влияющие на возможность конвертации (например, коллцентр или шоурум).

Анализ зависимостей

Распределение сессий рекламного траффика по месяцам



Первый месяц наблюдения за рекламным трафиком является неинформативным из-за малого количества сессий.

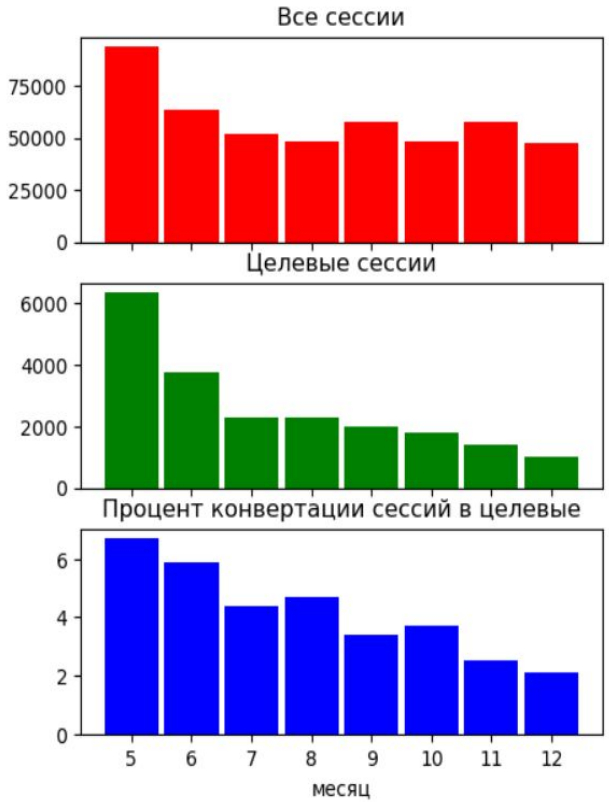
В июле наблюдалось значительное падение всех показателей. Виной может являться “форсмажор” в отделе, ответственном за рекламу в соцсетях. В причинах стоит разобраться.

В остальных месяцах количество сессий в целом росло, но процент конвертации находился на одном уровне.

Процент конвертации рекламы в соцсетях уступает проценту конвертации по всему датасету (который равен 2,9%) в среднем около 1%. Есть над чем поработать.

Анализ зависимостей

Распределение сессий органического траффика по месяцам



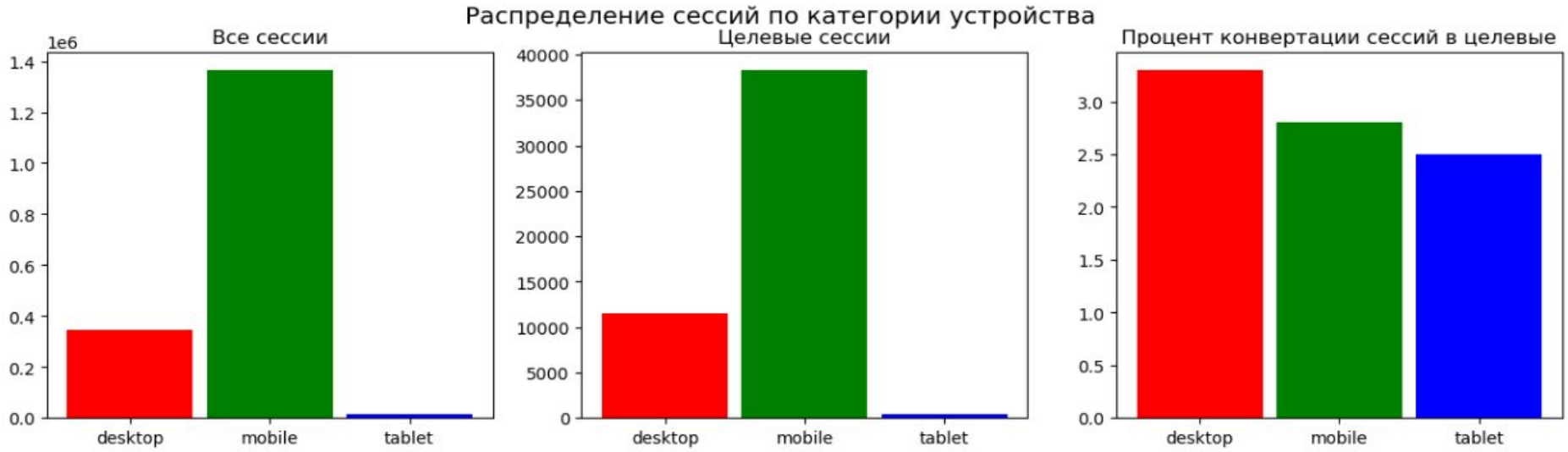
В первом месяце наблюдения все показатели были на отличном уровне.

Во втором месяце показатели еще оставались на хорошем уровне.

Затем, если общее количество сессий колебалось в одном диапазоне, то по целевым сессиям и проценту конвертации наблюдался постепенный спад.

В итоге в конце отчетного периода процент конвертации снизился до уровня ниже среднего по датасету. В причинах стоит разобраться.

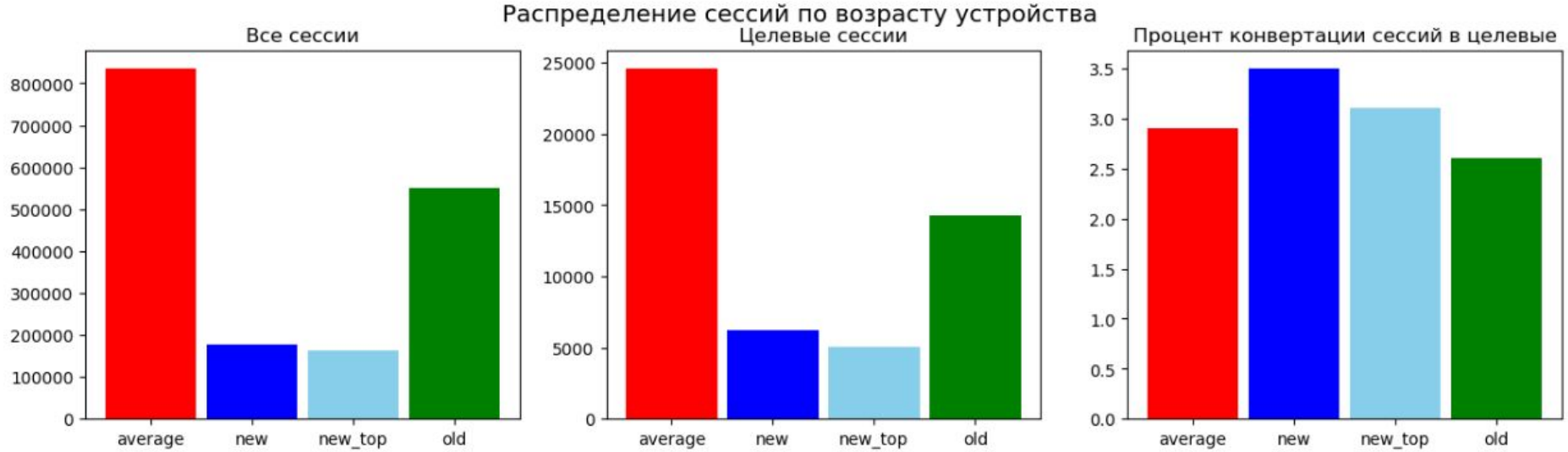
Анализ зависимостей



Подавляющее большинство сессий генерируются мобильными устройствами, что объясняется их гораздо большей распространенностью.

Но процент конверсии выше со стационарных пк. Объяснение: клиенты обращаются с рабочих мест, работа с пк более вдумчивая, удобная, эргономичная.

Анализ зависимостей

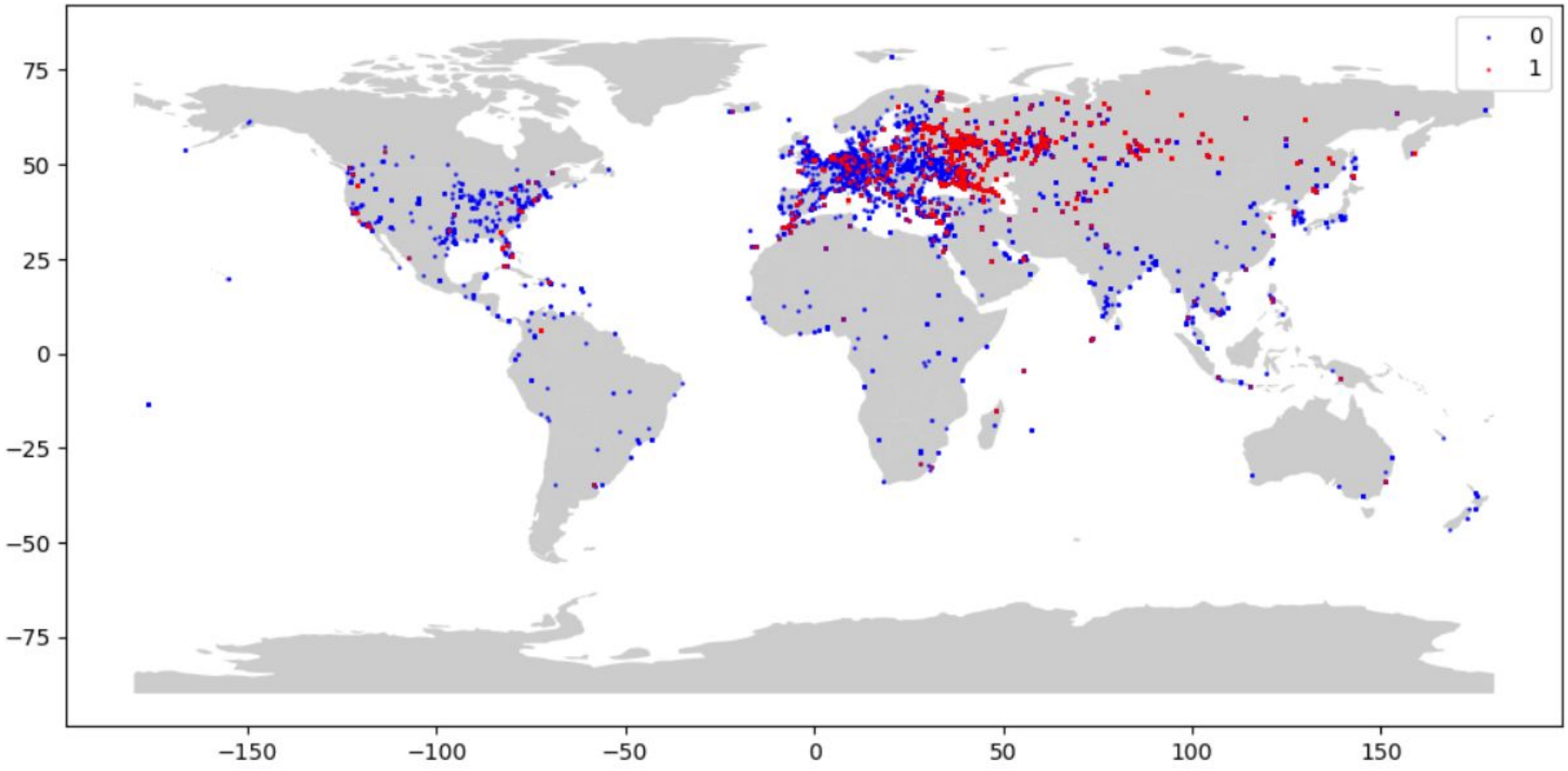


Новые устройства уступают как по количеству всех сессий, так и по количеству целевых, но процент конвертации с них выше.

Объяснением может быть следующее: новые устройства имеют преимущество в быстродействии, размере, эргономичности, разрешении и качестве экрана.

Также обладатель более нового устройства, как правило более платежеспособен, что повышает процент конверсии.

Зависимость от географических координат



Наиболее крупные “очаги” скопления сессий - европейская часть России и Европа.

Нормализация и преобразование категориальных фичей

```
# Типы данных колонок.
df.dtypes
```

```
visit_number          int64
device_category      object
device_os            object
geo_country          object
target                int64
utm_source_upd       object
utm_medium_upd       object
utm_campaign_upd     object
utm_adcontent_upd    object
device_brand_upd     object
device_browser_upd   object
month                 int64
day_of_month          int64
day_of_week           int64
hour_of_day           int32
screen_width          int32
screen_height         int32
device_age           object
social_net_adv        int64
organic_traffic       int64
is_post_soviet        int64
is_europe             int64
is_near_east          int64
city_lat            float64
city_long           float64
```

Количество уникальных значений в категориальных колонках

```
device_category        3
device_os             12
geo_country           66
utm_source_upd        57
utm_medium_upd        25
utm_campaign_upd     118
utm_adcontent_upd     72
device_brand_upd      51
device_browser_upd    14
device_age             4
```

Применяем StandarDScaler и OneHotEncoder к соответствующим признакам.

Удаляем колонки, которые были исходными для формирования признаков.

Имеем финальный датафрейм, готовый к моделированию, следующей размерности

```
df.shape
```

```
(1728226, 437)
```

Моделирование. LogisticRegression

```
# Инициализируем фичи и целевую переменную.
x = df.drop(['target'], axis=1)
y = df.target

# Делим данные на трейн и тест.
# Тесту выделили 20% (держим в уме последующий делёж трейна на 'фолды' в кросс-валидации)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=73)
```

Сейчас и в будущем для воспроизводимости результатов зафиксируем random_state=73. Число Шелдона Купера)

```
# Проводим кросс-валидацию тренировочной выборки, разделив ее на четыре 'фолда'.
cv_score = cross_val_score(logreg, x_train, y_train, scoring='roc_auc', cv=4, n_jobs=-1

# Средний показатель roc_auc и его дисперсия.
print(round(cv_score.mean(),4), round(cv_score.std(),4))
```

С гиперпараметрами по умолчанию

```
0.6819 0.0012
```

Путем перебора допустимых комбинаций значений параметров solver (значения: lbfgs, liblinear, newton-cg, newton-cholesky, sag, saga) и penalty (значения: l2, l1, none) установлено, что изменние метрики происходит незначительное.
Также незначительный прирост дает тюнинг параметра “C”, отвечающего за степень регуляризации.

```
logreg = LogisticRegression(C=0.1, penalty='l2', solver='sag', random_state=73, n_jobs=-1)
logreg.fit(x_train, y_train)
logreg_pred_test = logreg.predict(x_test)
logreg_probs = logreg.predict_proba(x_test)[:, 1]
print('Итоговая тюнингованная модель Логистической регрессии')
print('Значение метрики на тестовой выборке: ', round(roc_auc_score(y_test, logreg_probs),4))
```

```
Итоговая тюнингованная модель Логистической регрессии
Значение метрики на тестовой выборке:  0.6844
```

Моделирование. RandomForestClassifier

```
# Проводим кросс-валидацию тренировочной выборки, разделив ее на четыре 'фолда'.
cv_score = cross_val_score(forest, x_train, y_train, scoring='roc_auc', cv=4, n_jobs=-1

# Средний показатель roc_auc и его дисперсия.
print(round(cv_score.mean(),4), round(cv_score.std(),4))
```

С гиперпараметрами по умолчанию

```
0.6341 0.0013
```

Так как подбор гиперпараметров путем перекрестной кросс-валидации невозможен ввиду ограниченности ресурсов пк и большого размера датасета, их подбор будем осуществлять поэтапно путем перебора в цикле нескольких значений определенного параметра.

```
# Тюнинг глубины дерева.
max_depth_grid = [15, 20, 25, 30, 35]
# Минимальное количество объектов для ращепления
min_samples_split_grid = [2, 3, 5, 7, 9]
```

```
# Тюнинг количества деревьев в лесу.
trees_grid = [80, 120, 140, 160, 180]
# Тюнинг числа признаков, по которым ищется разбиение
max_features_grid = [25, 30, 35, 40, 45]
```

```
# Тюнинг минимального числа листьев в вет
min_samples_leaf_grid = [1, 2, 3, 5, 7]
```

```
forest= RandomForestClassifier(max_features=35, min_samples_split=5, min_samples_leaf=2, n_estimators=180, max_depth=25,
                               random_state=73, n_jobs=-1)
forest.fit(x_train, y_train)
forest_pred_test = forest.predict(x_test)
forest_probs = forest.predict_proba(x_test)[:, 1]
print('Итоговая тюнингованная модель Случайного леса')
print('Значение метрики на тестовой выборке: ', round(roc_auc_score(y_test, forest_probs),4))
```

Обучим модель с лучшими параметрами на тренировочной выборке.
Выведем значение roc_auc на тестовой (ранее отложенной) выборке.

```
Итоговая тюнингованная модель Случайного леса
Значение метрики на тестовой выборке:  0.7111
```

Моделирование. MLPClassifier

```
# Проводим кросс-валидацию тренировочной выборки, разделив ее на четыре 'фолда'.
cv_score = cross_val_score(neural, x_train, y_train, scoring='roc_auc', cv=4, n_jobs=-1

# Средний показатель roc_auc и его дисперсия.
print(round(cv_score.mean(),4), round(cv_score.std(),4))
```

С гиперпараметрами по умолчанию

```
0.6974 0.0015
```

```
# Тюнинг размера скрытого слоя.
hidden_layer_sizes_grid = [(100,), (150,), (200,), (250,), (300,)]
# Попробуем добавить дополнительный скрытый слой.
hidden_layer_sizes_grid = [(100,50), (150,75), (200,100), (250,125)]
# Тюнинг функции активации скрытого слоя.
activation_grid = ['identity', 'logistic', 'tanh', 'relu']
# Тюнинг коэффициента регуляризации с функцией активации 'logistic'
alpha_grid = [0.001, 0.01, 0.1, 1]
# Тюнинг коэффициента регуляризации с функцией активации 'tanh'
alpha_grid = [0.001, 0.01, 0.1, 1]
```

Подбор гиперпараметров осуществляем также поэтапно путем перебора в цикле нескольких значений.

```
neural = MLPClassifier(alpha=0.001, activation='tanh', hidden_layer_sizes=(100,50), random_state=73)
neural.fit(x_train, y_train)
neural_pred_test = neural.predict(x_test)
neural_probs = neural.predict_proba(x_test)[:, 1]
print('Итоговая тюнингованная модель Нейронной сети')
print('Значение метрики на тестовой выборке: ', round(roc_auc_score(y_test, neural_probs),4))
```

```
Итоговая тюнингованная модель Нейронной сети
Значение метрики на тестовой выборке:  0.6988
```

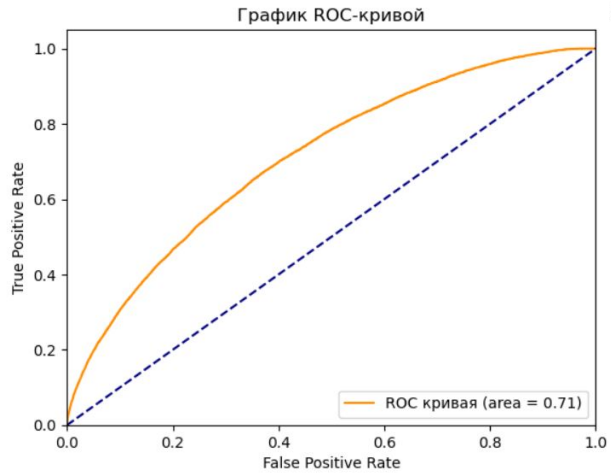
Моделирование. Итоги

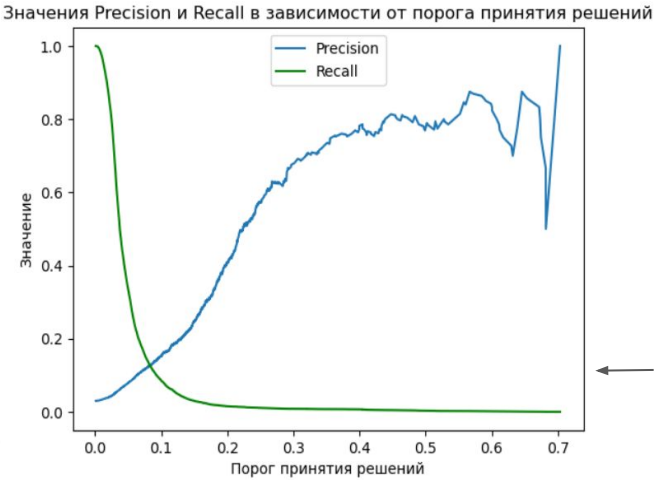
	RandomForestClassifier	LogisticRegression	MLPClassifier
среднее значение roc_auc на кросс-валидации при параметрах по умолчанию на train-выборке	0.6341	0.6819	0.6974
среднее значение roc_auc на кросс-валидации после тюнинга параметров на train-выборке	0.7070	0.0682	0.7002
дисперсия roc_auc на кросс-валидации после тюнинга параметров на train-выборке	0.0006	0.0010	0.0012
roc_auc после тюнинга на отложенной test-выборке	0.7111	0.6840	0.6988

```
# Обучаем лучшую модель на всём датасете.
x = df.drop(['target'], axis=1)
y = df.target

forest= RandomForestClassifier(max_features=35, min_samples_split=5, min_samples_leaf=2, n_estimators=180, max_depth=25,
                               random_state=73, n_jobs=-1)
forest.fit(x, y)
```

ПОБЕДИТЕЛЬ - СЛУЧАЙНЫЙ ЛЕС






```
# Сохраняем обученную модель в pickle-файл
filename = 'final_model.pickle'

with open(filename, 'wb') as file:
  pickle.dump(forest, file)
```

В зависимости от задач и рисков бизнесса при выборе порога принятия решений модели можно руководствоваться следующим графиком