



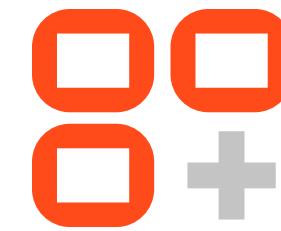
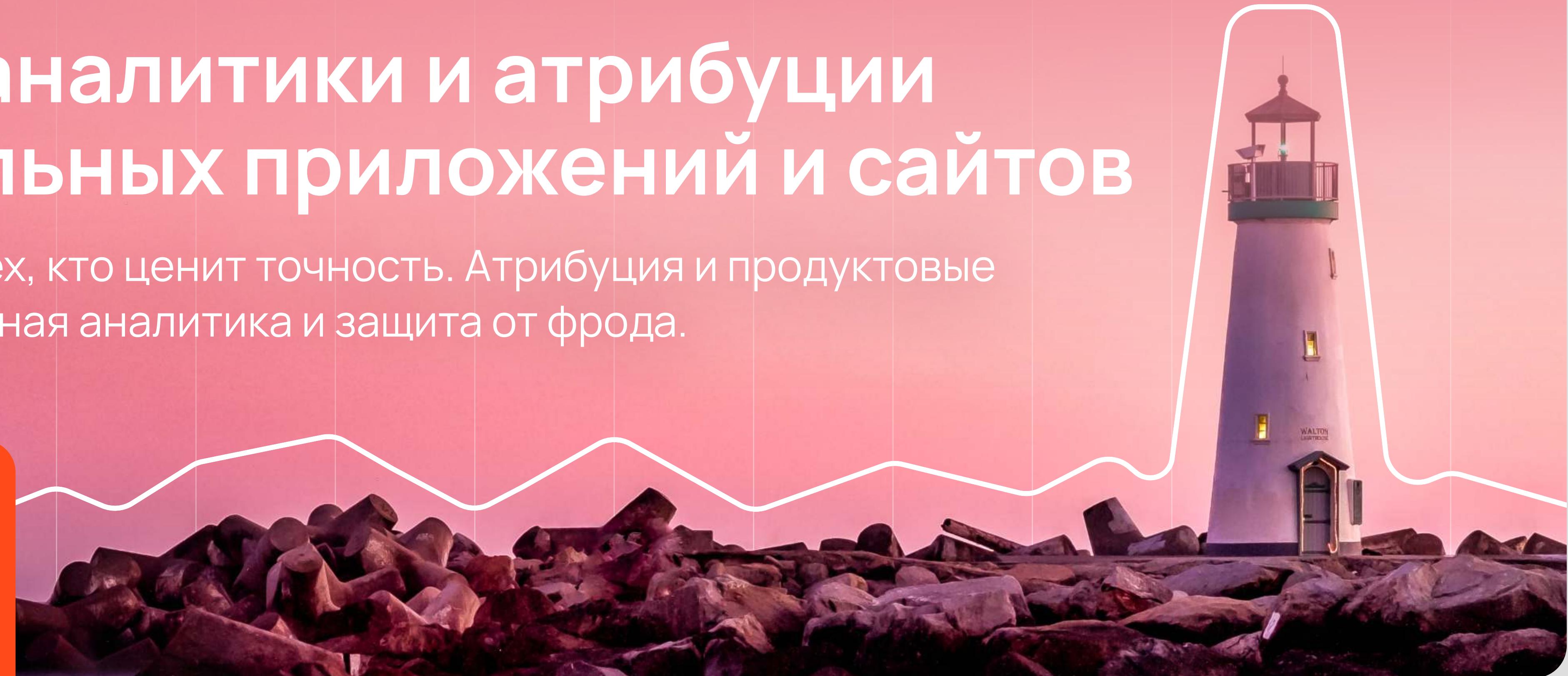
# Как самостоятельно построить прогноз LTV мобильного приложения

# Содержание

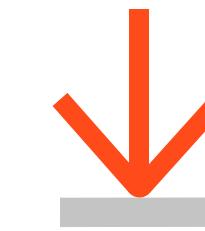
- 5 Введение
- 9 Постановка задачи и выбор метрик
- 13 Подготовка окружения, выбор моделей для использования
- 15 Применение модели catboost и линейной регрессии на приложении с большим количеством данных
- 23 Применение линейной модели
- 27 Применение коэффициентной и экстраполяционной модели на приложении с маленьким количеством данных
- 34 Применение коэффициентной модели на приложении с маленьким количеством данных
- 35 Экстраполяционная модель (логарифм)
- 41 Как улучшить модели
- 43 Заключение

# Система аналитики и атрибуции для мобильных приложений и сайтов

Инструменты для тех, кто ценит точность. Атрибуция и продуктевые метрики, предиктивная аналитика и защита от фрода.



Более 3 тысяч  
приложений



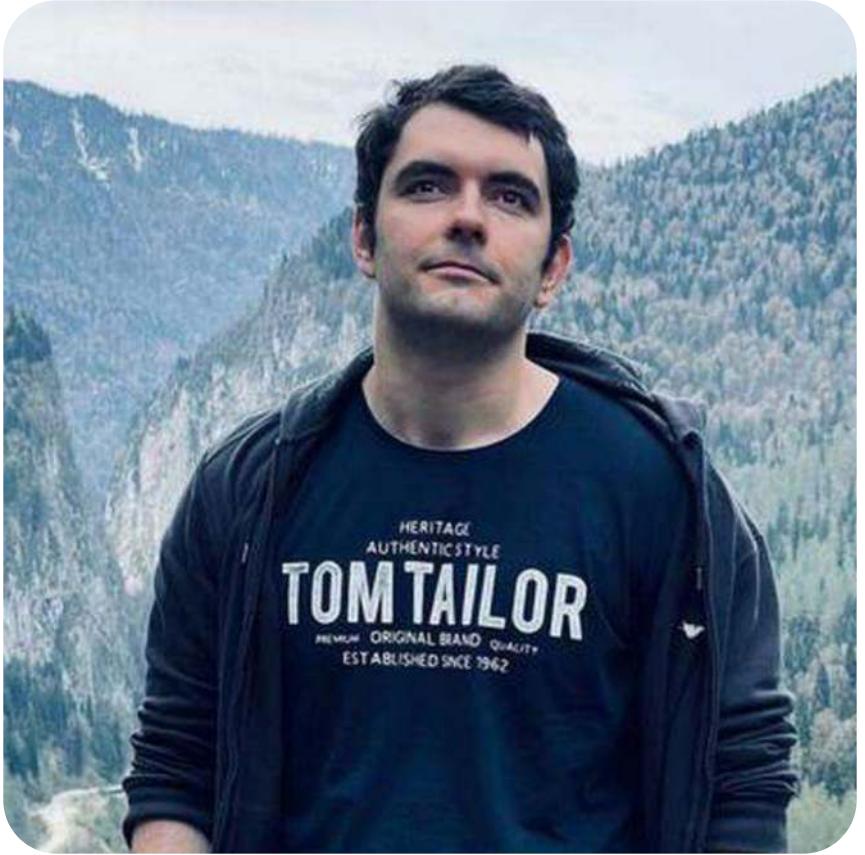
Более 3 млн новых  
установок ежедневно



Более 20 млрд  
передаваемых событий  
ежедневно

# Руководство по прогнозированию дохода по устройствам при наличии данных

Составлено специалистами MyTracker в 2022 году



**Хапкин Артем**

Программист-исследователь,  
Команда предиктивной аналитики



**Александр Смирнов**

Программист-исследователь,  
Команда предиктивной аналитики



# Введение

Если вы занимаетесь закупкой трафика, то вам нужны инструменты, которые помогут вам составить промежуточную оценку вложенных в рекламу средств и иметь возможность перенаправить бюджеты или отключить рекламу.

Одним из таких инструментов являются модели по прогнозу LTV (дохода от пользователей) на требуемый горизонт. Обычно в качестве горизонта выбирают одно из следующих значений: 30 дней, 90 дней, 180 дней, год или два года.



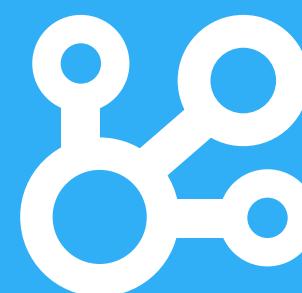


# Зачем нужно это руководство

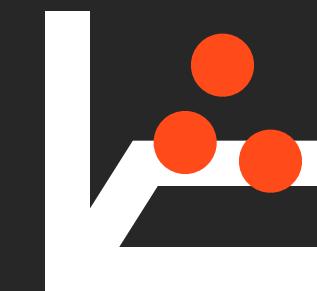


Прогнозирование дохода приложения – это сложно. Оно сопряжено с трудностями в предобработке и анализе данных, а также с выбором подходящих моделей и правильных метрик.

Поэтому в этом руководстве мы проанализируем и предложим различные подходы к прогнозированию дохода – рассмотрим разные модели для разных кейсов:



Модель  
на основе catboost



Модель на основе  
линейной регрессии



Коэффициентная  
модель



Модель на основе  
экстраполяции  
логарифмом

# Чему вы научитесь

Работать с сырыми  
данными,  
предобращивать их



Представлять  
в виде пригодных  
данных для анализа



Подбирать модели  
и корректно оценивать  
их по метрикам



# Параметры построения прогноза

## Мы будем строить прогноз по устройствам

Прогнозы можно строить в разрезе устройств или пользователей – подходы очень похожи. Для этого руководства мы выбрали прогноз по устройствам.

## Мы будем строить прогноз на горизонт в 180 дней

Потому что в индустрии чаще смотрят именно на окупаемость через 180 дней. Но для любых других горизонтов логика останется точно такой же.

## Рассматриваемый нами период накопления данных – 8 дней

Прогнозы можно строить в разрезе устройств или пользователей – подходы очень похожи. Для этого руководства мы выбрали прогноз по устройствам.

## Вводные данные для прогноза – платежи и вид трафика

Платежи – это обязательные данные, без которых нельзя построить прогноз дохода. А тип трафика – это дополнительные данные, которые мы взяли для примера.

# Постановка задачи и выбор метрик

Значение LTV для устройств вычисляется с момента установки приложения на устройство. Также возможно строить модели для прогноза по пользователям. Но в этом руководстве будут рассмотрены только прогнозы по устройствам.

## Оптимальный период исторических данных для построения прогноза

Прогноз LTV строится по устройствам, на которые уже установили приложение. Таким образом, у нас есть возможность некоторое время понаблюдать за жизнью этого устройства и собрать ряд признаков для прогнозирования.

Есть две крайности, которые могут повлиять на качество прогноза:

- **Слишком большой период накопления данных.** Нет смысла копить данные в течение всех 180 дней. Прогноз нужен для принятия решений об отключении неэффективных источников трафика и перенаправлении бюджетов. Если затянуть со сбором данных, то время для принятия решений будет упущено.
- **Слишком малый период накопления данных.** Если не ждать вовсе и начать собирать данные уже через час после установки, то можно не собрать достаточно большое количество информации для качественного прогнозирования. В результате, могут быть приняты ошибочные решения.

В индустрии период для сбора данных варьируется от 3 до 15 дней.

**Мы рекомендуем собирать данные в течение 8 дней.**

Потому что с одной стороны, уже можно принимать решение на основе полученных прогнозов и перенаправлять бюджеты, а с другой – есть время на сбор признаков, которые обеспечат качество прогноза.

## Данные для построения прогноза

Для построения прогноза необходимы данные о платежах с устройства – это главная вводная, без которой нельзя спрогнозировать доход.

Дополнительно можно использовать информацию о запусках и сессиях, соцдем-признаки и информацию о партнере и кампании, которая привела устройство.

В наших примерах мы будем использовать данные о платежах и типе трафика: органический или платный.

# Как оценить качество прогнозной модели

При закупке трафика важнее получить точный прогноз дохода с когорты, нежели с одного отдельного устройства (за редким исключением, когда работа трафик-менеджера нацелена на вылавливание китов).

Когорта – это набор пользователей, обладающих одним общим свойством. Например, это могут быть пользователи, установившие приложение 1 января 2022 года. Или все пользователи из России, установившие приложение в декабре 2021 года.

Для оценки работы прогнозной модели мы будем использовать **метрику когортной ошибки**. Это функция, которая позволяет рассчитать точность модели предсказания для выбранной когорты. Для ее расчета нужно сложить фактическое LTV всех устройств, входящих в когорту. И так же рассчитать прогнозное LTV когорты, как сумму всех прогнозов, входящих в когорту.

$$\left| \frac{pLTV}{LTV} - 1 \right|$$

$pLTV$  – прогнозное LTV когорты  
 $LTV$  – фактическое LTV когорты

В индустрии чаще всего используют два типа когорт – **дневную** (устройства, установившие приложение в один день) и **месячную** (устройства, установившие приложение в один месяц).

При использовании модели полезно оценивать ошибку на наборе когорт. Например, когда наблюдение за моделью продолжается месяц и для каждой дневной когорты посчитана своя ошибка.

Также, в прогнозах можно использовать **среднее значение**. Однако часто встречается ситуация, когда в разных когортах наблюдаются разные значения фактического LTV. Так, например, если дневная ошибка на дне с фактическим LTV в 1000 составляет 50%, то среднее значение не отразит эту разницу. А меньшая ошибка на большем значении фактического LTV важнее. Для решения этой проблемы следует использовать средневзвешенную ошибку - когортную ошибку по фактическому значению LTV.

$$\frac{\sum_{i=1}^n \left| \frac{pLTV_{cohort_i}}{LTV_{cohort_i}} - 1 \right| \cdot LTV_{cohort_i}}{\sum_{i=1}^n LTV_{cohort_i}}$$

$pLTV_{cohort_i}$  – прогноз по  $i$ -ой когорте  
 $LTV_{cohort_i}$  – фактический доход  $i$ -ой когорты

Именно **средневзвешенные ошибки по дневным и месячным когортам** мы будем использовать в дальнейшем для оценки качества прогнозной модели.

# Подготовка окружения и данных, выбор моделей для использования

Всего мы рассмотрим 4 варианта моделей прогнозирования LTV

- модель на основе catboost
- модель на основе линейной регрессии
- коэффициентная модель
- модель на основе экстраполяции логарифмом

Для начала работы нам нужно импортировать следующие библиотеки, классы и функции

Чтобы сделать примеры более наглядными, мы будем работать с тестовыми данными. Вы можете [скачать их](#) из Облака.

```
import warnings
warnings.filterwarnings("ignore")

import sys
import pandas as pd
import numpy as np
from datetime import timedelta
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from datetime import datetime, timedelta
from matplotlib import pyplot as plt
from plotly import graph_objs as go
from dateutil.relativedelta import relativedelta
import warnings
import shap
```

# Работа с тестовыми данными

По ссылке [в Облаке](#) есть два файла – `big_data.csv` и `small_data.csv`

## **`big_data.csv`**

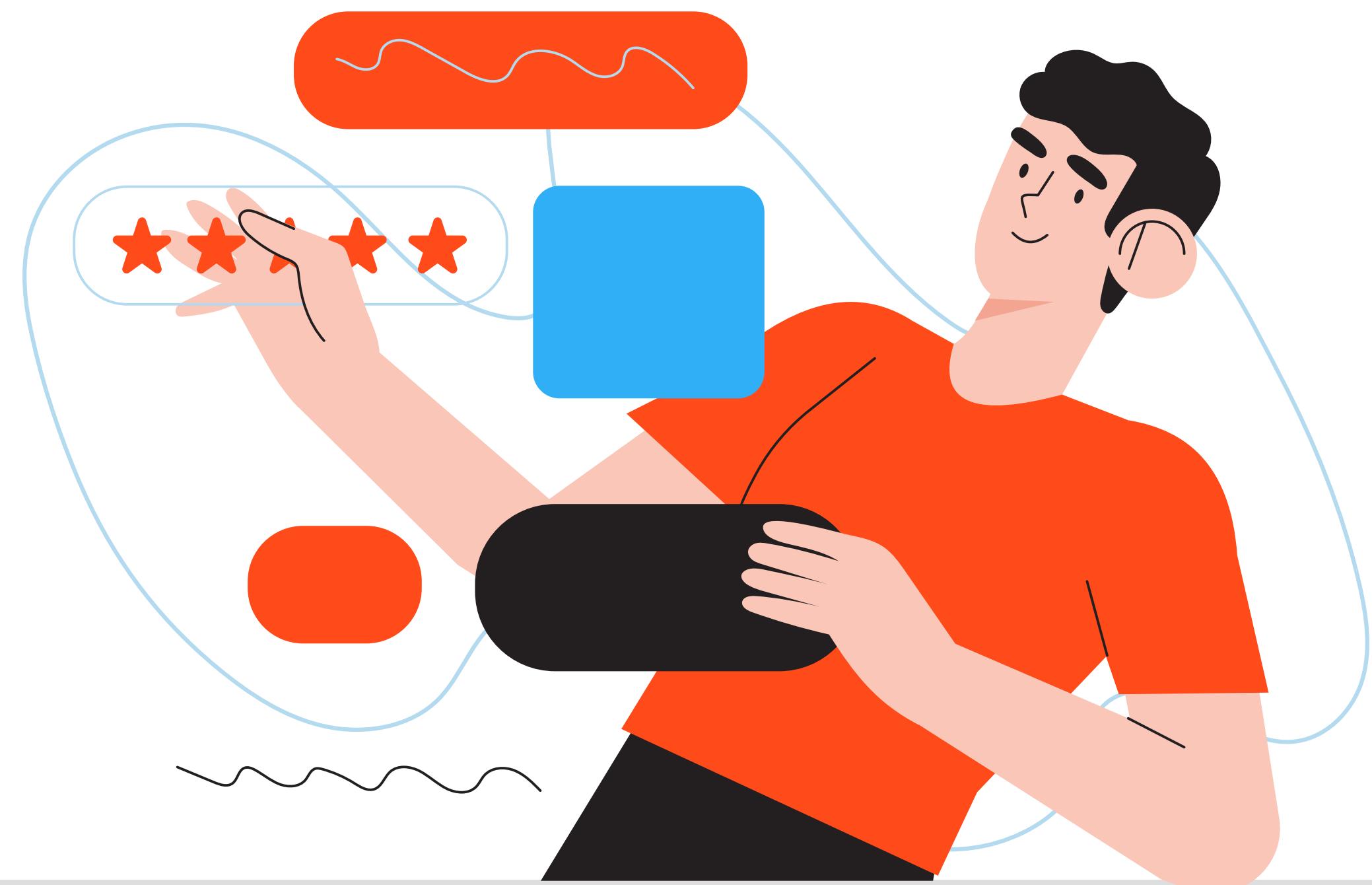
В этом файле содержатся данные о сырых транзакциях устройств для большого приложения. В нем есть следующие поля:

- `idCountry` – страна пользователя
- `dtInstall` – дата установки
- `dtevent` – дата платежа
- `idTrafficType` – тип трафика, с которого была выполнена установка
- `sumUsd` – размер платежа в долларах
- `device_id` – id устройства

## **`small_data.csv`**

В этом файле содержатся данные о кумулятивном (накопленном) LTV устройств. В нем есть следующие поля:

- `install_date` – дата установки
- `ltvDay1`, ..., `ltvDay8` – накопленный LTV с 1 по 8 день жизни
- `device_id` – id устройства



# Применение модели catboost и линейной регрессии на приложении с большим количеством данных

Предположим, что в вашем приложении есть список транзакций за период с 2021-01-01 по 2021-10-28. Посмотрим на данные. Для этого загрузим файл big\_data.csv.

## Просмотр данных

```
df_raw = pd.read_csv("big_data.csv")
print("Dataset size: ", df_raw.shape)
print("Min date: ", df_raw.dtInstall.min())
print("Max date: ", df_raw.dtInstall.max())
```

Размер датасета: (914744, 6)  
Минимальная дата: 2021-01-02  
Максимальная дата: 2021-10-29

```
print("Data preview:")
df_raw.head()
```

## Предпросмотр данных

	<b>idCountry</b>	<b>dtInstall</b>	<b>dtEvent</b>	<b>idTrafficType</b>	<b>sumUsd</b>	<b>device_id</b>
0	76	2021-10-16	2022-04-13	1	18.91	device_0000001
1	202	2021-10-28	2022-05-03	1	2.85	device_0000002
2	202	2021-10-28	2022-04-04	1	3.57	device_0000002
3	81	2021-10-02	2022-04-06	2	28.27	device_0000003
4	81	2021-10-02	2022-04-03	2	2.90	device_0000003

Каждая строчка – это платеж устройства. Также в строке есть сумма платежа в долларах, дата установки, дата платежа, тип трафика и страна.

Введем дополнительный столбец, который показывает, сколько дней прошло от даты установки до даты платежа. В дальнейшем он нам потребуется.

```
df_raw['timedelta'] = (pd.to_datetime(df_raw.dtEvent) - pd.to_datetime(df_raw.dtInstall)).apply(lambda x: x.days)
```

# Преобразование данных и подготовка для обучения

В виде сырых транзакций признаки плохо подходят для анализа. Сделаем дополнительные колонки кумулятивного LTV с 1 по 8 день наблюдения за устройством.

```
df_to_work = df_raw.groupby('device_id')['dtInstall', 'idCountry',  
'idTrafficType'].first().reset_index()  
  
def get_sum_ltv_column(df_raw, df_to_work, ltv_day_list=[1,2,3,4,5,6,7,8,180]):  
    for ltv_day in ltv_day_list:  
        buff_df = df_raw[df_raw.timedelta <= ltv_day]  
        buff_df = buff_df.groupby('device_id')['sumUsd'].sum().reset_index()  
        buff_df.rename(columns={'sumUsd': f'ltvDay{ltv_day}'}, inplace=True)  
        df_to_work = df_to_work.merge(buff_df, how='left', on='device_id').fillna(0)  
  
    return df_to_work  
  
df = get_sum_ltv_column(df_raw, df_to_work)  
df.head()
```

	device_id	dtInstall	idCountry	idTrafficType	ltvDay1	ltvDay2	ltvDay3	ltvDay4	ltvDay5	ltvDay6	ltvDay7	ltvDay8	ltvDay180
0	device_0000001	2021-10-16	76	1	0.0	0.0	0.0	0.0	0.0	0.0	12.39	15.88	83.89
1	device_0000002	2021-10-28	202	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	17.96
2	device_0000003	2021-10-02	81	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	255.41
3	device_0000004	2021-10-29	67	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	57.82
4	device_0000005	2021-10-22	26	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	22.63

При построении модели важно учитывать, какой процент устройств платит только на 8 день после установки. На этапе прогнозирования нет возможности узнать их признаки, а на фактическое LTV они влияют.

Посмотрим на процент устройств, заплативших к восьмому дню хотя бы один раз. Именно у них будут ненулевые полезные признаки.

```
print(df[df.ltvDay8 > 0].shape[0] / df.shape[0] * 100)
```

60.96103849852202

Разделим данные на трейн и тест. Очень важно проводить разделение так, чтобы во время теста были известны все данные, которые используются в тренировочном наборе.

Если прогнозной модели нужно знать LTV 180 дня, то для обучения можно использовать только те когорты, которые установили приложение раньше, чем за 180 дней до момента прогнозирования.

Рассмотрим на примере. Пусть тестовый период будет в следующих датах:

```
start_date_test = '2021-09-21'  
end_date_test = '2021-10-21'
```

Вычислим период, который можно использовать для обучения.

```
fit_available_date = pd.Timestamp(start_date_test) - pd.Timedelta(days=180)  
fit_available_date
```

```
Timestamp('2021-03-25 00:00:00')
```

Можно использовать данные раньше 2021-03-25. Возьмем для обучения следующий 3-х месячный период:

```
start_date_fit = '2020-12-25'  
end_date_fit = '2021-03-25'
```

Модель будет работать только с теми устройствами, у которых есть платежи.

```
df = df[df['ltvDay180'] > 0]
```

```
df_fit = df[(df.dtInstall >= start_date_fit) & (df.dtInstall <= end_date_fit)]  
df_test = df[(df.dtInstall >= start_date_test) & (df.dtInstall <= end_date_test)]
```

# Начинаем обучать модель catboost

Подход к обучению будет следующим: 6 раз разобьем данные для обучения на тренировочный и валидационный набор, как при кросс-валидации. На каждом разбиении подберем параметры модели, используя валидационной набор. Обучим модели с найденными параметрами на каждом разбиении.

У нас получится 6 моделей. Итоговым прогнозом для устройства будет среднее прогнозов этих моделей. Таким образом, прогноз будет более устойчив к выбросам.

```
catboost_params = {'iterations': 900,
                   'learning_rate': 0.05,
                   'depth': 5,
                   'custom_metric': ['MAE'],
                   'random_seed': 63,
                   'early_stopping_rounds': 50,
                   'use_best_model': True}

categorical_features = ['idCountry', 'idTrafficType']
ltv_features = ['ltvDay1', 'ltvDay2', 'ltvDay3', 'ltvDay4', 'ltvDay5', 'ltvDay6', 'ltvDay7', 'ltvDay8']
features = categorical_features + ltv_features
categorical_features_index = [0, 1]

last_day = 8
target = ['ltvDay180']

models_list = []
random_state_list = [2, 4, 5, 90, 3567, 180]
for random_state in random_state_list:

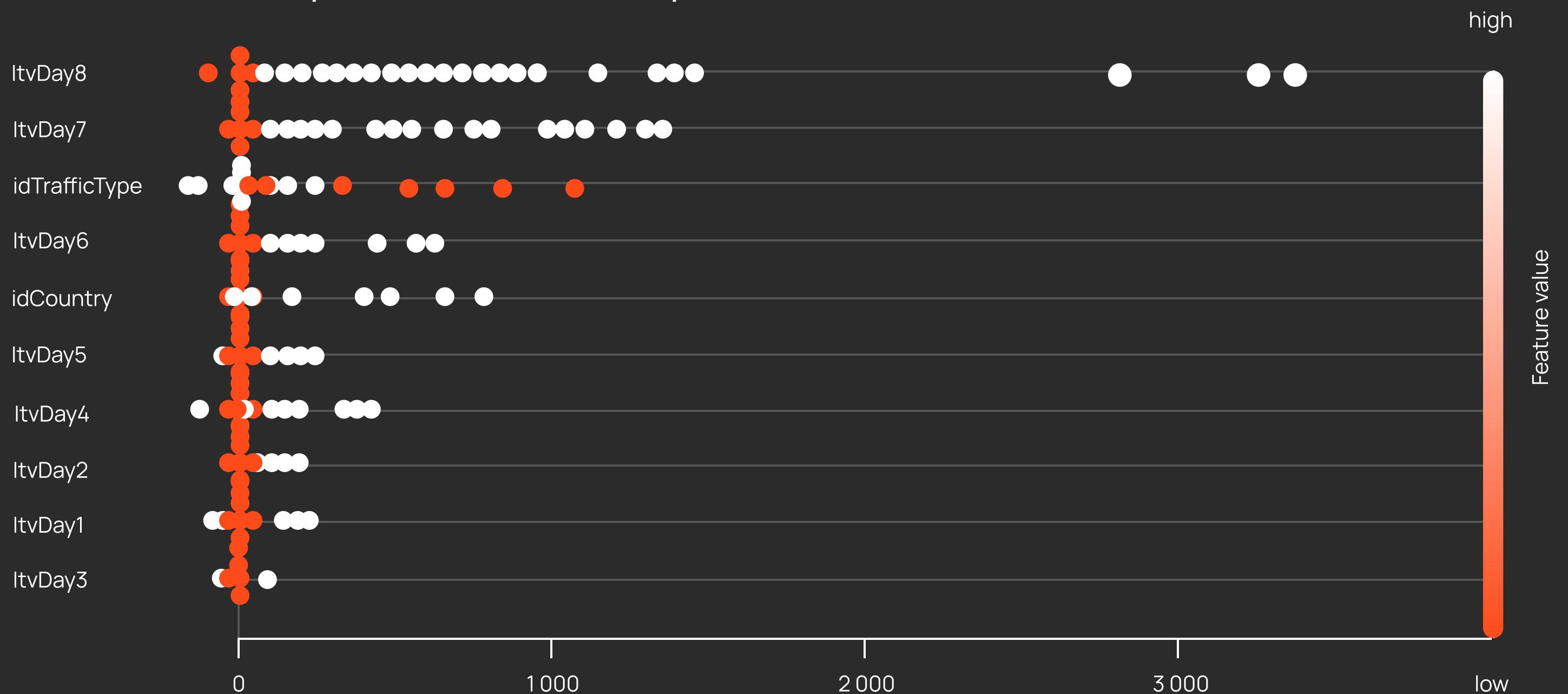
    X_train, X_valid, y_train, y_valid = train_test_split(df_fit[features],
                                                          df_fit[target],
                                                          test_size=0.2,
                                                          random_state=random_state)

    model = CatBoostRegressor(**catboost_params)
    model.fit(X_train.astype('int'),
              y_train.astype('int'),
              cat_features=categorical_features_index,
              eval_set=(X_valid.astype('int'), y_valid.astype('int')),
              silent=True)
    models_list.append(model)
```

Рассмотрим одну из моделей и попробуем оценить важность ее признаков с помощью SHAP.

```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_train)
shap.summary_plot(shap_values, X_train)
```

SHAP value (impact on model output)



Чем активнее признак подкрашен белым или оранжевым, тем больше влияния он имеет на финальный прогноз модели.

Видно, что самыми важными признаками тут является ItvDay8. Это неудивительно. Ведь на этот день аккумулируется максимально доступная информация по платежам пользователей.

Модель может давать прогноз, который меньше LTV 8 дня, поэтому необходимо ограничить возможный прогноз для каждого устройства фактическим для него значением LTV на 8 день.  
Далее получим данные с прогнозами:

```
df_test['predict_catboost'] = 0
for model in models_list:
    df_test['predict_catboost'] += model.predict(df_test[features])
df_test['predict_catboost'] /= len(random_state_list)
df_test['predict_catboost'] = round(df_test['predict_catboost'].clip(df_test[f'ltvDay{last_day}']),2)
df_test.head()
```

	device_id	dtInstall	id Country	idTraffic Type	ltvDay1	ltvDay2	ltvDay3	ltvDay4	ltvDay5	ltvDay6	ltvDay7	ltvDay8	ltvDay180	predict_catboost
0	device_0000001	2021-10-16	76	1	0.00	0.00	0.00	0.00	0.00	0.00	12.39	15.88	83.89	101.07
2	device_0000003	2021-10-02	81	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	255.41	64.05
7	device_0000008	2021-10-07	200	1	111.42	111.42	111.42	166.63	166.63	166.63	166.63	166.63	2789.79	764.21
8	device_0000009	2021-10-02	67	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	61.46	77.96
9	device_00000010	2021-10-12	76	1	25.83	25.83	51.28	51.28	51.28	337.78	388.66	414.10	1196.51	2042.60

# Применение линейной модели

Теперь рассмотрим другой подход. Будем использовать простую линейную регрессию на признаках кумулятивных платежей. Также строим 6 моделей по разным разбиениям данных. Потом, при прогнозировании, будем брать среднее среди них.

```
ridge_list = []
for random_state in random_state_list:

    X_train, X_valid, y_train, y_valid = train_test_split(df_fit[features],
                                                          df_fit[target],
                                                          test_size=0.2,
                                                          random_state=random_state)

    model = Ridge(fit_intercept=False)
    model.fit(X_train[ltv_features].astype('int'),
              y_train.astype('int'))
    ridge_list.append(model)

df_test['predict_ridge'] = 0
for model in ridge_list:
    df_test['predict_ridge'] += model.predict(df_test[ltv_features].astype('int')).reshape(-1)
df_test['predict_ridge'] /= len(random_state_list)
df_test['predict_ridge'] = round(df_test['predict_ridge'].clip(df_test[f'ltvDay{last_day}']), 2)
```

Рассмотрим значения коэффициентов каждого признака у последней модели.

```
model_feature_coef = {item[0]: item[1] for item in zip(ltv_features, list(model.coef_[0]))}
for key, value in model_feature_coef.items():
    print(f'{key}: {value}')
```

```
ltvDay1: -0.5729238765495388
ltvDay2: 0.49663237909267643
ltvDay3: -1.411067907152598
ltvDay4: -0.5692373059294994
ltvDay5: -0.0987175550386999
ltvDay6: -2.554070323527861
ltvDay7: 3.431151127557224
ltvDay8: 4.252497903631954
```

Видно, что, как и у catboost, самая важная фича – кумулятивный платеж к 8-му дню.

# Применение функции подсчета ошибки

Чтобы посчитать метрики по формулам, в начале напишем функцию подсчета ошибок.

```
def get_errors(df, predict_columns=[],  
              target_column=['LtvDay180'],  
              date_column='dtInstall',  
              weighted[]):  
  
    if len(predict_columns) == 0:  
        raise Exception("Zero len of predict columns! Can't calculate any error")  
  
    columns = predict_columns + target_column  
  
    # compute common errors  
    if len(weighted) == 0:  
        df_error = df.groupby(date_column[0])[columns].sum().reset_index()  
        for col in predict_columns:  
  
            error = np.abs((df_error[col] / df_error[target_column[0]]) - 1)  
            df_error[col + '_error'] = error  
  
    df_error.drop(columns, axis=1, inplace=True)  
    return df_error
```

```
# compute weighted errors  
else:  
    groupby_cols = [date_column[0]] + weighted  
    df_error = df.groupby(groupby_cols)[columns].sum().reset_index()  
    drop_columns = target_column  
  
    # DataFrame for errors  
    buff_df = pd.DataFrame()  
    buff_df[date_column[0]] = sorted(df[date_column[0]].unique())  
    for col in predict_columns:  
        df_error[col + '_error'] = np.abs((df_error[col] / df_error[target_column[0]]) - 1)  
  
        df_error['mult_' + col] = df_error[col + '_error'] * df_error[target_column[0]]  
  
    temp_df = df_error.groupby(date_column[0])[target_column[0],  
                                'mult_' + col].sum().reset_index()  
  
    temp_df[col + '_error_weighted'] = temp_df['mult_' + col]/temp_df[target_column[0]]  
  
    buff_df = buff_df.merge(temp_df[[date_column[0], col + '_error_weighted']],  
                           how='left', on=date_column[0])  
  
return buff_df
```

# Оценка качества моделей, подсчет метрик

Посчитаем дневные ошибки для линейной модели и модели catboost.

```
predict_columns = ['predict_catboost', 'predict_ridge']
df_day_errors = get_errors(df_test, predict_columns=predict_columns)
```

Отрисуем ошибки:

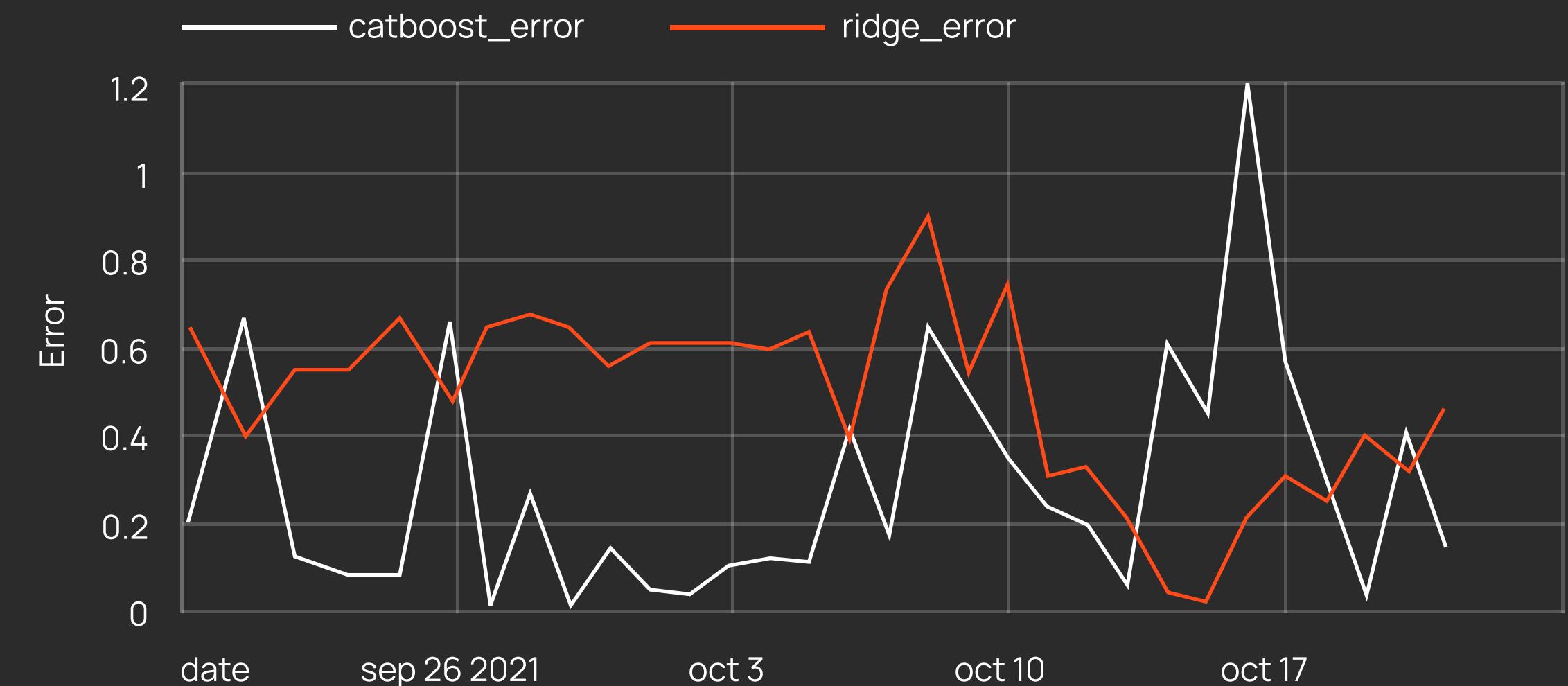
```
layout = go.Layout()
fig = go.Figure()

# two curves
for col in predict_columns:
    name = col.split('_')[1] + '_error'
    fig.add_trace(go.Scatter(x=df_day_errors.dtInstall, y=df_day_errors[col + '_error'],
name=name))

# figure layout
fig.update_layout(
    title_text="Day-app errors catboost and ridge.",
    title_font_size=20,
    yaxis_title='Error',
    xaxis_title='date'
)
fig
```

```
# figure layout
fig.update_layout(
    title_text="Day-app errors catboost and ridge.",
    title_font_size=20,
    yaxis_title='Error',
    xaxis_title='date'
)
fig
```

Дневные ошибки для линейной модели и модели catboost



Рассчитаем средневзвешенную ошибку по типам трафика:

```
predict_columns = ['predict_catboost', 'predict_ridge']
df_day_traf_errors = get_errors(df_test, predict_columns=predict_columns,
weighted=['idTrafficType'])
```

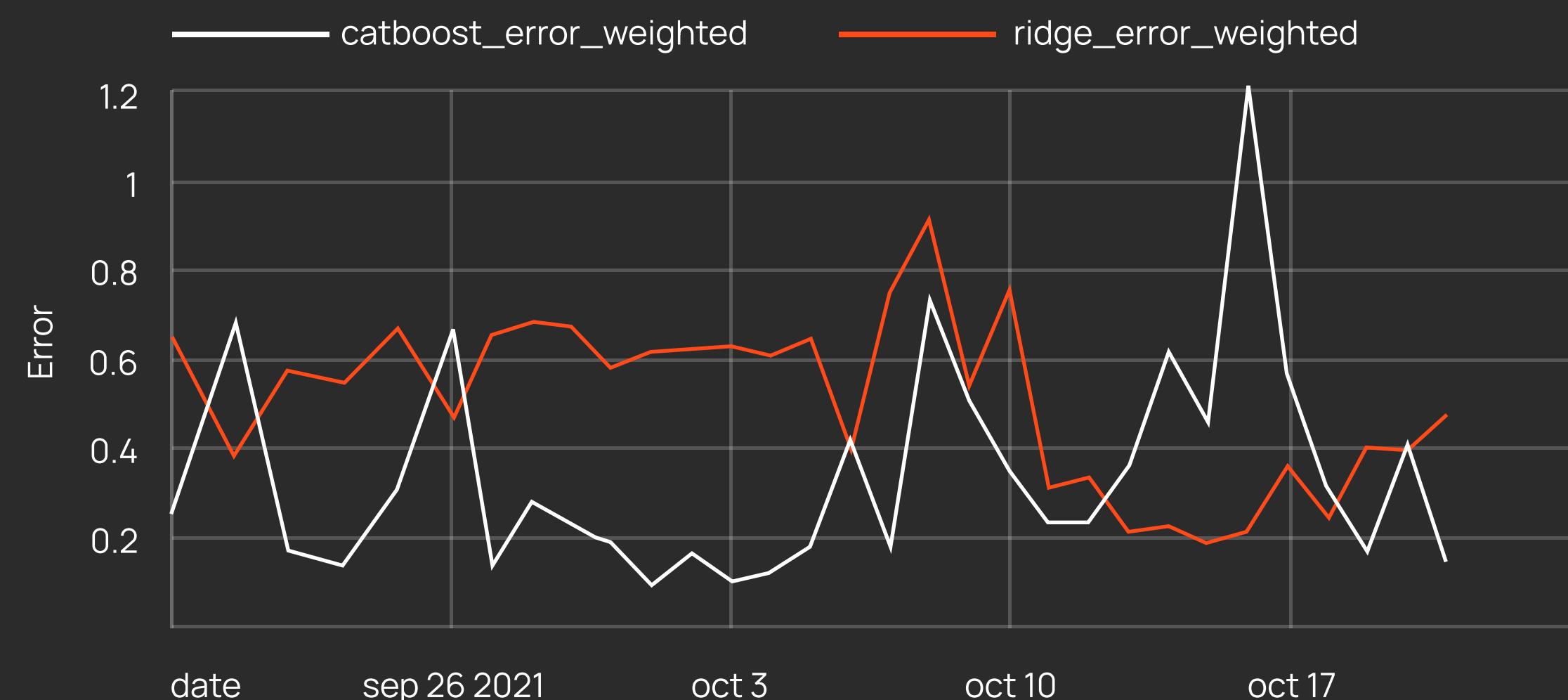
Отрисуем ошибку:

```
layout = go.Layout()
fig = go.Figure()

# two curves
for col in predict_columns:
    name = col.split('_')[1] + '_error_weighted'
    fig.add_trace(go.Scatter(x=df_day_traf_errors.dtInstall,
                            y=df_day_traf_errors[col + '_error_weighted'], name=name))

# figure layout
fig.update_layout(
    title_text="Day-app-traff errors catboost and ridge.",
    title_font_size=20,
    yaxis_title='Error',
    xaxis_title='date'
)
```

Дневные ошибки для линейной модели и модели catboost



df\_day\_traf\_errors.mean()

```
predict_catboost_error_weighted 0.343704
predict_ridge_error_weighted 0.506448
dtype: float64
```

# Применение коэффициентной и экстраполяционной модели на приложении с маленьким количеством данных

**Рассмотрим приложение, которое по объему меньше предыдущего.**

Разберем два кейса: когда есть данные за 180 дней, и когда есть данные только за последний месяц с небольшим, но хочется получить прогноз на 180 дней.

**Для первого кейса** будет использоваться коэффициентная модель, которая основана на подсчете относительных коэффициентов.

**Для второго** – попытаемся получить функцию, которая максимально похожа на долгосрочное поведение пользователей.

Так как мы берем данные за прошедший период, то можно посчитать реальную ошибку. Однако если в реальности нет больших исторических данных, то можно не считать ошибку на далекий горизонт, а использовать сразу эти подходы.

# Просмотр данных

Для демонстрации модели используем данные из файла small\_data.csv. В нем уже сделана подготовка кумулятивных платежей к наблюдаемым дням. Поэтому строки представляют из себя информацию об устройстве: день установки, признаки кумулятивных платежей.

```
test_dataframe = pd.read_csv("small_data.csv")
test_dataframe.head()
```

	dtInstall	ItvDay1	ItvDay2	ItvDay3	ItvDay4	ItvDay5	ItvDay6	ItvDay7	ItvDay8	ItvDay180	device_id
0	2020-12-29	4.841925	4.841925	4.841925	4.841925	4.841925	4.841925	4.841925	4.841925	4.841925	device_0000001
1	2021-07-04	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	734.917738	device_0000002
2	2021-01-24	0.000000	10.768286	10.768286	10.768286	25.232335	25.232335	25.232335	25.232335	25.232335	device_0000003
3	2021-09-03	3.559892	3.559892	3.559892	3.559892	3.559892	3.559892	3.559892	3.559892	3.559892	device_0000004
4	2021-04-17	450.986491	450.986491	1356.347486	1356.347486	1356.347486	1853.196859	1853.196859	1853.196859	2615.129891	device_0000005

Период данных:

```
test_dataframe.install_date.min(), test_dataframe.install_date.max()
('2020-12-03', '2021-10-01')
```

## Написание вспомогательных классов

Прогнозирование коэффициентной моделью осуществляется индивидуально для каждого заплатившего устройства в рамках дневной когорты. Прогноз строится как умножение признака `ltvDay8` на специальный коэффициент. Варианты его расчета будут приведены ниже.

Для коэффициентной модели необходимы знания о признаке кумулятивного платежа за 180 дней `ltvDay180` по истории. Поэтому важно правильно подбирать период для тренировочных данных. Для этого напишем класс, который для каждого дня установки будет из имеющихся данных выбирать правильную обучающую и тестовую выборку.

Составление обучающей выборки: допустим, есть дата установки, на которую необходимо сделать прогноз — `prediction_date`. Для создания обучающей выборки берутся все установки с признаками, описанными выше, которые были в период `[prediction_date - gap - train_days, prediction_date - gap]`. Гар выбирается таким образом, чтобы на обучающем наборе были известны все необходимые данные.

```
class Splitter:
    def __init__(self, df, train_days, gap):
        self.train_days = train_days
        self.gap = gap
        self.df = df
        self.date_format = "%Y-%m-%d"

    def get_train(self, date):
        date_dt = datetime.strptime(date, self.date_format)
        start_train = (date_dt - timedelta(days=self.gap +
                                             self.train_days)).strftime(self.date_format)
        end_train = (date_dt - timedelta(days=self.gap)).strftime(self.date_format)
        df = self.df[(self.df["install_date"] < end_train) &
                     (self.df["install_date"] >= start_train)].copy()
        return df

    def get_predict(self, date):
        df = self.df[self.df["install_date"] == date].copy()
        return df
```

Коэффициентная модель использует исторические данные об отношениях кумулятивных платежей за 180 дней к кумулятивным платежам за 8 дней как тренировочную выборку для расчета специального коэффициента.

## Основные параметры коэффициентной модели:

- 1 Размер обучающей выборки `train_days` (число дней, которые будут взяты для обучения).
- 2 Горизонт прогнозирования `ltv` (в нашем примере равен 180 дням). Именно его необходимо использовать в качестве отступа для выбора тренировочной выборки.

Стабильным вариантом расчета специального коэффициента является следующий подход: для каждого дня обучающей выборки строится отношение суммарных кумулятивных платежей за 180 дней к суммарным кумулятивным платежам за 8 дней. Специальный коэффициент - медиана этих отношений за все дни обучающей выборки. Вот класс с реализацией этой модели:

```
class CoefModelMedianDate:
    def __init__(self, train_days=30, lt=180, pred_day=8):
        self.train_days = train_days
        self.lt = lt
        self.pred_day = pred_day
        self.coef = None

    def fit(self, df):
        df_coef = df.groupby("install_date")[[f"ltvDay{self.lt}", f"ltvDay{self.pred_day}"]].sum()
        df_coef["coef"] = df_coef[f"ltvDay{self.lt}"] / df_coef[f"ltvDay{self.pred_day}"]
        self.coef = np.median(df_coef["coef"])

    def predict(self, df):
        if not self.coef:
            raise ValueError("Model is not fitted.")
        df_predict = df.copy()
        df_predict["prediction"] = df[f"ltvDay{self.pred_day}"] * self.coef
        return df_predict

Рассмотрим пример для даты 2022-07-01, создадим модель и сплиттер:
model = CoefModelMedianDate(
    train_days=30,
    lt=180,
)

splitter = Splitter(
    df=test_dataframe,
    train_days=model.train_days,
    gap=model.lt
)
```

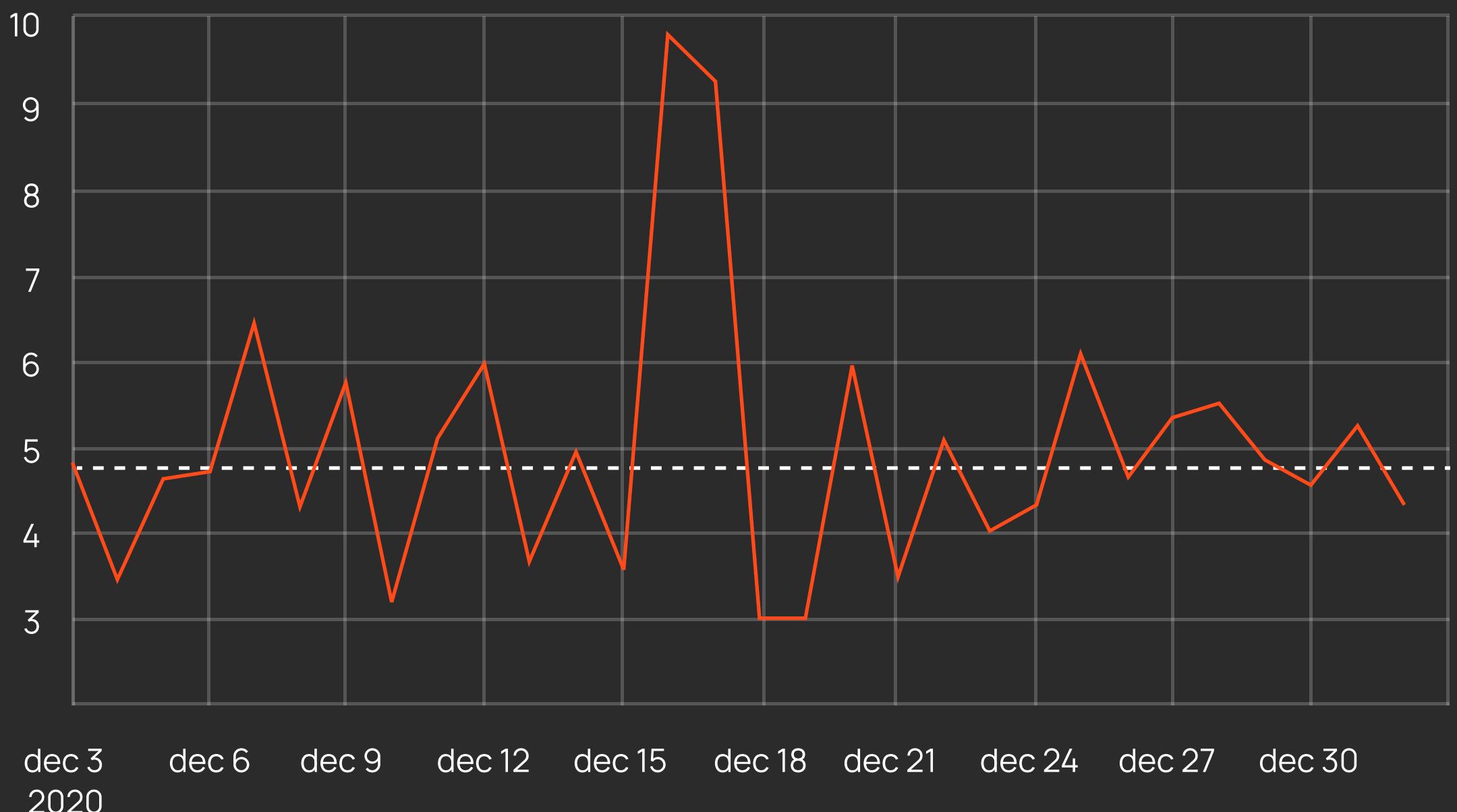
Выделим обучающий и тестовый датасеты:

```
df_train = splitter.get_train("2021-07-01")
df_predict = splitter.get_predict("2021-07-01")
```

Построим коэффициенты отношения по дням обучающей выборки, найдем медиану и отрисуем на графике:

```
data = []
df_coef = df_train.groupby("install_date")[[f"ltvDay{model.lt}",
f"ltvDay{model.pred_day}"]].sum()
df_coef["coef"] = df_coef[f"ltvDay{model.lt}"] / df_coef[f"ltvDay{model.pred_day}"]
data.append(go.Scatter(
    x=df_coef.index,
    y=df_coef["coef"],
    name="ltvDay180 / ltvDay8"
))
layout = go.Layout(title="Coefs on train dataset. ltvDay180 for ltvDay8.")
fig = go.Figure(data=data, layout=layout)
fig.add_hline(y=np.median(df_coef["coef"]), line_width=3, line_dash="dash", line_color="green",
annotation_text="median")
fig
```

Коэффициенты на обучающей выборке.  
Отношение LtvDay180 к LtvDay8



Этот же коэффициент получим обучением модели:

```
model.fit(df_train)
```

```
model.coef
```

4.847753039842198

Совершим прогноз для одного дня:

```
df_ready_predict = model.predict(df_predict)
```

```
df_ready_predict.head()
```

	device_id	dtInstall	id Country	idTraffic Type	ItvDay1	ItvDay2	ItvDay3	ItvDay4	ItvDay5	ItvDay6	ItvDay7	ItvDay8	ItvDay180	predict_catboost
0	device_0000001	2021-10-16	76	1	0.00	0.00	0.00	0.00	0.00	0.00	12.39	15.88	83.89	101.07
2	device_0000003	2021-10-02	81	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	255.41	64.05
7	device_0000008	2021-10-07	200	1	111.42	111.42	111.42	166.63	166.63	166.63	166.63	166.63	2789.79	764.21
8	device_0000009	2021-10-02	67	1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	61.46	77.96
9	device_00000010	2021-10-12	76	1	25.83	25.83	51.28	51.28	51.28	337.78	388.66	414.10	1196.51	2042.60

# Применение коэффициентной модели на приложении с маленьким количеством данных

Предыдущий пример был только для одного дня. Модель простая, поэтому для нее лучше использовать как можно более актуальные данные.

Напишем класс для переобучения модели и построения прогноза для каждого дня установки:

```
class Test:
    def __init__(self, df, model, start_date, end_date):
        self.df = df
        self.model = model
        self.start_date = start_date
        self.end_date = end_date
        self.splitter = Splitter(self.df, train_days=self.model.train_days, gap=self.model.lt)

    def run(self):
        dates = pd.date_range(self.start_date, self.end_date)
        df_res_list = []
        for prediction_date in dates:
            prediction_date = prediction_date.strftime(self.splitter.date_format)
            train_dataframe = self.splitter.get_train(prediction_date)
            predict_dataframe = self.splitter.get_predict(prediction_date)
            self.model.fit(train_dataframe)
            df_res_list.append(self.model.predict(predict_dataframe))
        df_res = pd.concat(df_res_list)
        return df_res
```

Для применения модели на периоде достаточно указать начало и конец этого периода, какая модель будет использоваться, а также данные с признаками устройств:

```
test = Test(
    df=test_dataframe,
    model=model,
    start_date="2021-07-01",
    end_date="2021-10-01"
)
df_coef_res = test.run()
```

Итоговое предсказание на 3-месячном периоде:

`df_coef_res.head()`

	dtInstall	ItvDay1	ItvDay2	ItvDay3	ItvDay4	ItvDay5	ItvDay6	ItvDay7	ItvDay8	ItvDay180	device_id	prediction
688	2021-07-01	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	366.891604	366.891604	6076.100262	device_0000689	1778.599887
1670	2021-07-01	183.329019	237.535335	237.535335	237.535335	237.535335	237.535335	237.535335	237.535335	237.535335	device_0001671	1151.512644
1985	2021-07-01	34.895094	34.895094	34.895094	34.895094	34.895094	34.895094	34.895094	34.895094	1365.075556	device_0001986	169.162799
2145	2021-07-01	0.000000	0.000000	0.000000	0.000000	19.643077	19.643077	19.643077	19.643077	32.167385	device_0002146	95.224786
2699	2021-07-01	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	9.500825	9.500825	9.500825	device_0002700	46.057651

## Подсчет качества коэффициентной модели

Используем известную нам функцию и рассчитаем среднюю ошибку за эти 3 месяца для модели:

```
predict_columns = ['prediction']
df_day_errors = get_errors(df_coef_res, predict_columns=predict_columns,
                           date_column=['install_date'],
                           target_column=['ItvDay180'])
df_day_errors.mean()
```

```
prediction_error 0.300865
dtype: float64
```

# Экстраполяционная модель (логарифм)

Для работы этой модели мы предполагаем, что платежи связаны с днем логарифмической зависимостью. Пользователи с момента установки приложения (особенно игрового) с течением времени будут пользоваться этим приложением все меньше и меньше. И платить меньше.

Поэтому для модели подойдет возрастающая функция, у которой с ростом аргумента (в нашем случае это день после установки), будет становиться меньше прирост значения. Логарифм этому свойству удовлетворяет, поэтому будем использовать его.

Однако есть возможность использовать и другие функции со степенью меньше единицы, например – квадратный корень.

Итоговый вид формулы расчета LTV на основе логарифма:

$$y = a \log(x + 1) + b$$

где  $x$  – день после установки приложения,

$y$  – сумма платежей ко дню  $x$ .

# Написание вспомогательных классов

Для применения модели необходимо найти коэффициенты а и b. Рассчитать значение на 180 день и на 8 день. А дальше использовать их отношение как специальный коэффициент для прогноза в коэффициентной модели. Для более точного вывода можно добавить LTV30 – так логарифм получится более точным, однако потребуется расширить отступ для получения тренировочной выборки.

Опишем класс с реализацией. Функцию экстраполяции по значению будем искать с помощью функции polyfit, библиотека numpy.

```
class ApprModelDateExtra:  
    def __init__(self, train_days=4, pred_day=8, lt=180, gap=0):  
        self.train_days = train_days  
        self.gap = 0  
        self.lt = lt  
        self.model = Ridge()  
        self.pred_day = pred_day  
        self.records = []  
  
    def fit(self, df):  
        df = df[df[f"ltvDay{self.pred_day}"] != 0]  
        df_train_list = []  
        for pred_day in list(range(1, self.pred_day + 1)):  
            df_tmp = pd.DataFrame()  
            df_tmp["y"] = df.groupby("install_date")[f"ltvDay{pred_day}"].sum()
```

```
df_tmp["log(x + 1)"] = np.log(pred_day + 1)  
df_train_list.append(df_tmp)  
df_train = pd.concat(df_train_list)  
X = df_train["log(x + 1)"].values  
y = df_train["y"]  
function = np.poly1d(np.polyfit(X, y, 1))  
self.coef = function(np.log(self.lt + 1)) / function(np.log(self.pred_day + 1))  
self.records.append(function)  
  
def predict(self, df):  
    if not self.coef:  
        raise ValueError("Model is not fitted.")  
    df_predict = df.copy()  
    df_predict["prediction"] = df[f"ltvDay{self.pred_day}"] * self.coef  
    return df_predict
```

Создадим экземпляр модели и проанализируем принцип работы на примере. Обратите внимание, что дар модели равен 0, потому что тренировочная выборка находится очень близко к текущему моменту. Что позволяет использовать модель при наличии небольшой истории:

```
extrapolation_model = ApprModelDateExtra(
    train_days=8,
    pred_day=8,
    lt=180
)
```

Создадим класс для получения тренировочной и тестовой выборки:

```
splitter_extrapolation = Splitter(
    df=test_dataframe,
    train_days=extrapolation_model.train_days,
    gap=extrapolation_model.gap
)
```

Возьмем тренировочную и тестовую выборку и продемонстрируем работу модели:

```
df_train = splitter_extrapolation.get_train("2021-07-01")
df_predict = splitter_extrapolation.get_predict("2021-07-01")
```

Так как `numpy.polyfit` умеет работать с линейными функциями, то для ее применения нам необходимо привести уравнение  $y=a\log(x+1)+b$  к линейному виду. Это просто сделать следующим образом:

$y=ax'+b$ , где  $x'=\log(x+1)$

Сформируем набор точек для последующей экстраполяции. Используем для этого все доступные нам кумулятивные платежи до 8 дня. Обратите внимание, что здесь мы используем суммарные кумулятивные платежи за дневные когорты:

```
df = df_train.copy()
df = df[df[f"ltvDay{extrapolation_model.pred_day}"] != 0]
df_train_list = []
for pred_day in list(range(1, extrapolation_model.pred_day + 1)):
    df_tmp = pd.DataFrame()
    df_tmp["y"] = df.groupby("install_date")[f"ltvDay{pred_day}"].sum()
    df_tmp["log(x + 1)"] = np.log(pred_day + 1)
    df_tmp["x"] = pred_day
    df_train_list.append(df_tmp)
df_extrapolation = pd.concat(df_train_list)
```

Получились следующие точки:

`df_extrapolation.head()`

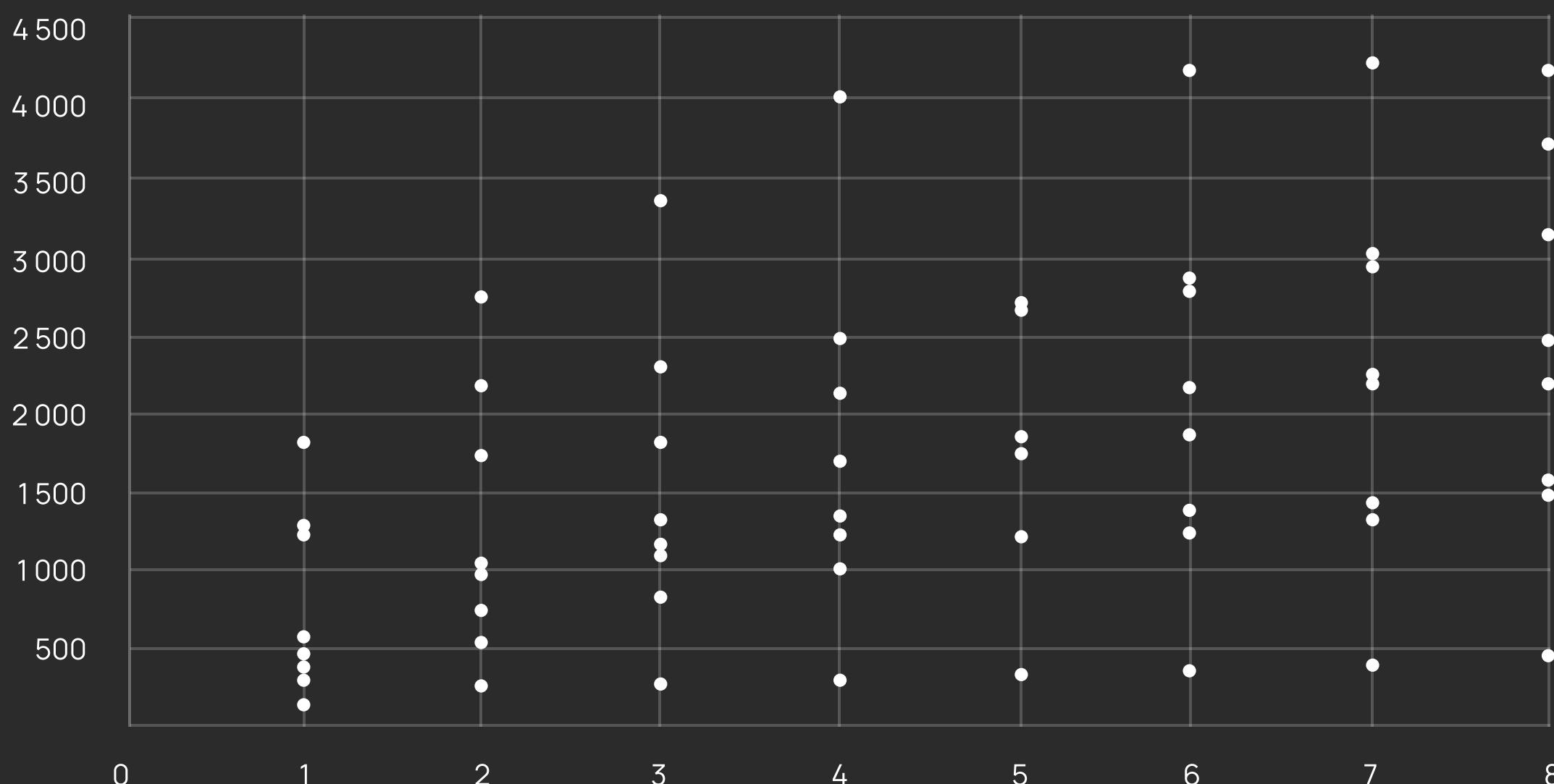
$y \ log(x+1) \ x$

install_date				
2021-06-01	295.672576	0.693147	1	
2021-06-01	1261.292849	0.693147	1	
2021-06-01	563.211456	0.693147	1	
2021-06-01	1234.928187	0.693147	1	
2021-06-01	1836.270119	0.693147	1	

Отрисуем их в виде графика:

```
data = []
data.append(go.Scatter(
    x=df_extrapolation["x"],
    y=df_extrapolation["y"],
    mode="markers"
))
layout = go.Layout(title="Dots to extrapolate.")
fig = go.Figure(data=data, layout=layout)
fig
```

Точки для построения экстраполяции



Теперь найдем экстраполяционную функцию. Помним, что она линейная:

```
X = df_extrapolation["log(x + 1)"].values
y = df_extrapolation["y"]
function = np.poly1d(np.polyfit(X, y, 1))
```

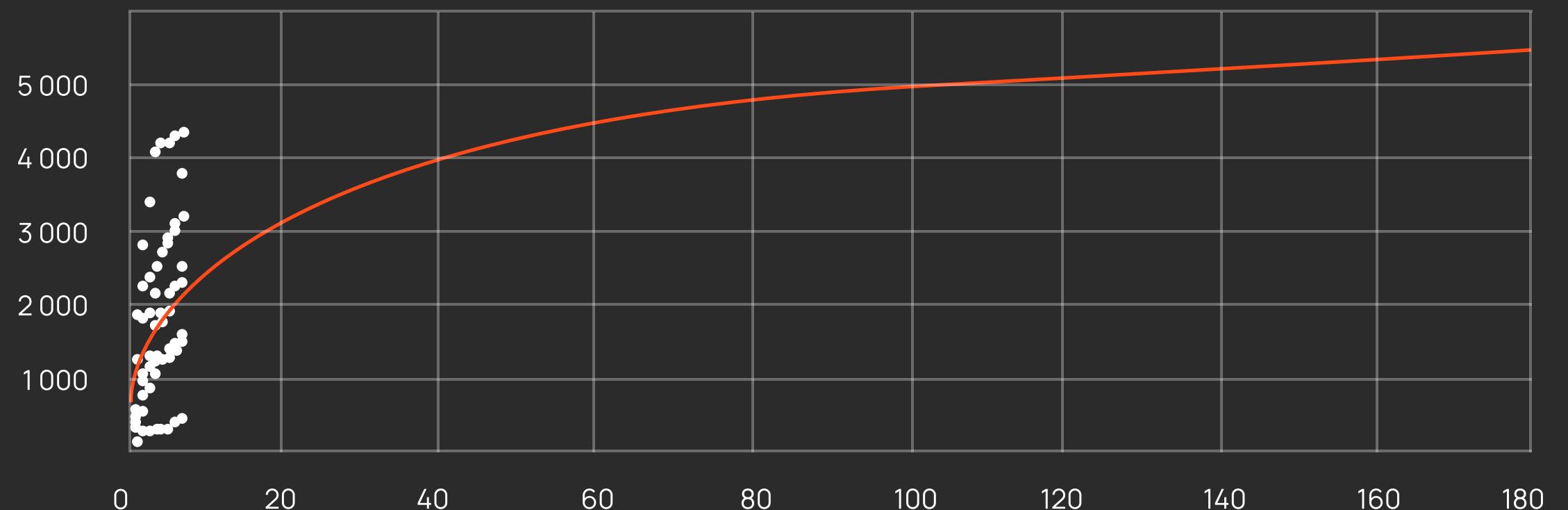
Отрисуем полученную функцию:

```
data = []
data.append(go.Scatter(
    x=df_extrapolation["x"],
    y=df_extrapolation["y"],
    mode="markers",
    name="extrapolation dots"
))
x = np.array((range(1, 180 + 1)))
y = function(np.log(x + 1))
data.append(go.Scatter(
    x=x,
    y=y,
    name="log"
))
layout = go.Layout(title="Dots to extrapolate")
fig = go.Figure(data=data, layout=layout)
fig
```

## Точки для построения экстраполяции

● extrapolation dots

— log



Итоговый коэффициент для прогноза:

`extrapolation_model.fit(df_train)`

`extrapolation_model.coef`

2.3251867333678455



# Применение экстраполяционной модели

Как и для коэффициентной модели, проведем прогнозирование на периоде с переобучением на каждый день:

```
test = Test(  
    df=test_dataframe,  
    model=extrapolation_model,  
    start_date="2021-07-01",  
    end_date="2021-10-01"  
)  
df_extrapolation_res = test.run()
```

## Подсчет качества модели экстраполяции

Используем известную функцию и рассчитаем среднюю ошибку за эти 3 месяца для модели:

```
predict_columns = ['prediction']  
df_day_errors = get_errors(df_extrapolation_res, predict_columns=predict_columns,  
                           date_column=['install_date'],  
                           target_column=['ltvDay180'])  
df_day_errors.mean()  
  
prediction_error 0.416167  
dtype: float64
```

# Как улучшить модели

## Более детальная работа с плательщиками и неплательщиками к 8-му дню по отдельности

Как в модели catboost, так и в других моделях, мы не разделяли плательщиков и неплательщиков к 8-му дню. Но практика показывает, что если работать с ними по отдельности (считать разными моделями, добавлять разные коэффициенты), то качество вырастает, в среднем, на 10-15%.

## Сравнение распределений на трейне и на teste

Если более внимательно подойти к сравнению распределений: посмотреть, как отличается платежное поведение пользователей на этих двух периодах, то можно скорректировать поведение двух моделей в лучшую сторону. Например, посмотреть новые предметы в корзине, как изменился средний чек и прочее.

## Использование более свежих данных

Мы отступили на полгода назад от начала тестовой выборки, чтобы набрать данные о поведении пользователей. При этом мы никак не использовали поведение пользователей, которые пришли месяц или два назад. А анализ их поведения будет более точен по отношению к новым пользователям, так как они стоят ближе. Это также позволяет достичь более высокого качества предиктов.

## Использование других моделей

Если catboost для одного приложения сработал хорошо, то это не значит, что для другого он будет также эффективен. Возможно, для другого приложения больше подойдет линейная модель или использование нейросетей. Это надо уметь считать и валидировать.

## Использование других данных

Если есть данные о сессиях, запусках, внутриигровых событиях, то их тоже нужно использовать. По статистике, модель от этого будет эффективнее на 5-10%.

## Ввод разных коэффициентов

Если считать не один коэффициент от 180-го к 8-му дню, то это тоже повысит качество прогноза. Например, можно считать коэффициенты доплаты или коэффициенты похожести выборок.

## Другие метрики для анализа

В определенных случаях имеет смысл больше анализировать не дневную ошибку, а недельные или месячные ошибки. На практике – чем крупнее срез, тем меньше будет ошибка.

# Заключение

**Существует много моделей для прогнозирования дохода мобильного приложения. И у каждой модели есть свои требования и ограничения.**

Поэтому подбор оптимальной модели может стать довольно непростой задачей. А на ручную настройку уйдет много времени и сил, особенно у человека без опыта.

Мы в MyTracker осознаем все сложности работы с данными, поэтому стремимся оптимизировать эти процессы. В наших автоматических прогнозах LTV учитываются особенности данных и их доступные объемы. Наши алгоритмы сами подбирают наиболее подходящую из перечисленных или других предиктивных моделей. Помимо этого, мы проводим ретроспективные тесты моделей. Все это позволяет получить эффективную модель с хорошей предсказательной способностью.

Средняя точность прогноза LTV в MyTracker – 80-90%. Узнайте, как сегодняшние решения повлияют на ваш бизнес через месяц, полгода или два года – запросите бесплатное демо предиктивной аналитики от MyTracker.

[Запросить демо](#)