Генерация синтетических данных на основе связанных таблиц

In [1]:
```
%load_ext watermark
```

In [2]:
```
%watermark
```

```
Last updated: 2023-07-06T16:40:45.040923+03:00

Python implementation: CPython
Python version       : 3.7.16
IPython version      : 7.34.0

Compiler    : Clang 14.0.0 (clang-1400.0.29.202)
OS          : Darwin
Release     : 22.5.0
Machine     : x86_64
Processor   : i386
CPU cores   : 8
Architecture: 64bit
```

In [3]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [4]:
```
import pandas as pd
import sqlalchemy
import multi_gen_func as mgf
from sdv.metadata import MultiTableMetadata
from sdv.multi_table import HMASynthesizer
from sdv.evaluation.multi_table import evaluate_quality, run_diagnostic
```

Глобальные ограничения

1. **Размер таблиц**. Чем длиннее и шире таблицы, тем больше ресурсов будет использоваться. В тестовом примере максимальный размер - 30000 строк и 11 полей (после коррекции связей между таблицами, подробнее в соответствующем блоке)
2. **Глубина связей**. Для оптимальной работы необходимо ограничиться вторым уровнем глубины, то есть table1->table2->table3, где **->** это связь типа *основная_таблица -> дочерняя*. При добавлении ссылки table3->table4 потребление ресурсов возрастает в разы

Базовая конфигурация

В данной секции необходимо определить базовую конфигурацию данных, с которыми идет работа. Далее подробнее о каждой ячейке

Первичные ключи

Конфиг первичных ключей - словарь типа {имя_таблицы: имя_поля }

Ограничение:

Данное решение запрещает использовать составные первичные ключи. В данном случае ключевые поля нужно свести к единственному (можно использовать хеш от значений данных полей)

In [5]:
```
pkeys = {'bookings': 'book_ref',
         'tickets': 'ticket_no',
         'ticket_flights': 'key',
         'flights': 'flight_id',
         'airports': 'airport_code',
         'aircrafts': 'aircraft_code',
         'seats': 'key'}
```

Регулярки. Опциональная ячейка

Если есть желание добавить регулярку, то можно создать словарь типа {имя_таблицы: {имя_поля: регулярка} }

In [ ]:
```
regex = {'flights': {'flight_no': 'PG[0-9]{5}',
                     'aircraft_code': '[0-9A-Z]{3}',
                     'arrival_airport': '[A-Z]{3}',
                     'departure_airport': '[A-Z]{3}',
                     'flight_id': '[0-9]{5}'},
         'bookings': {'book_ref': '[A-Z0-9]{5}'},
         'tickets': {'ticket_no': '[0-9]{10}',
                     'book_ref': '[A-Z0-9]{5}',
                     'passenger_id': '[0-9]{4} [0-9]{6}'},
         'ticket_flights': {'ticket_no': '[0-9]{10}',
                            'flight_id': '[0-9]{5}'},
         'airports': {'airport_code': '[A-Z]{3}'},
         'aircrafts': {'aircraft_code': '[0-9A-Z]{3}'},
         'seats': {'aircraft_code': '[0-9A-Z]{3}',
                   'seat_no': '[0-9A-K]'}
        }
```

Отношения и ключи

Здесь описываются все связи между таблицами

Словарь из кортежей типа:

{ (имя_родительской_таблицы, имя_дочерней_таблицы):
(первичный_ключ_родительской, внешний_ключ_дочерней) }

Ограничение

Если между таблицами имеется **более одной связи**, то для обхода уникальности ключей словаря необходимо добавить **третий элемент** в один из кортежей-ключей, пример с таблицами airports и flights ниже

In [6]:
```
relations_and_keys = {('bookings', 'tickets'): ('book_ref', 'book_ref'),
                      ('tickets', 'ticket_flights'): ('ticket_no', 'ticket_n
                      ('flights', 'ticket_flights'): ('flight_id', 'flight_i
                      ('airports', 'flights'): ('airport_code', 'departure_a
                      ('airports', 'flights', 1): ('airport_code', 'arrival_
                      ('aircrafts', 'flights'): ('aircraft_code', 'aircraft_
                      ('aircrafts', 'seats'): ('aircraft_code', 'aircraft_co
```

Дополнительно скрываемые поля

Пояснение **для категориальных или текстовых данных**: в пакете sdv существует тип id, который гарантирует обезличенность данных. По умолчанию данный тип получают все ключи, которые представляются в виде некой инкрементальной числовой или символьной последовательности (зависит от исходных данных). Для более реалистичного внешнего вида этим полям можно задать шаблон регулярного выражения для генерации (подобная ячейка находится выше).

В случае, если есть необходимость скрыть неключевые текстовые/категориальные поля, создается словарь типа {имя_таблицы: имя_поля }

In [7]:
```
other_important_fields = {'tickets': 'passenger_id',
                          'flights': 'flight_no',
                          'seats': 'seat_no'}
```

Кастомные типы

Здесь можно задать другие типы полей, исчерпывающий список по ссылкам ниже:

https://docs.sdv.dev/sdv/reference/metadata-spec/sdtypes#conceptual-sdtypes

https://faker.readthedocs.io/en/master/providers.html

Для использования достаточно добавить название типа (в случае остальных типов sdv) или просто название функции (в случае пакета Faker).

Итоговый объект - словарь типа {имя_таблицы: [имя_поля, тип] }

In [8]:
```
other_types_fields = {'tickets': ['passenger_name', 'name']}
```

Работа с SQL:

Если исходные данные тянутся из БД, то необходимо 3 объекта:

1. sqlalchemy engine
2. строка-название схемы в БД
3. sql-конфиг. О нем подробнее:

Создается словарь типа {имя_таблицы: аргументы}, где аргументом является либо **None**, либо **словарь**:

В случае **None** функция вытянет данные с аргументами по умолчанию, а именно - все поля и лимит 30000

Если нужны не все (или дополнительные поля), а также больше/меньше строк, их можно передать в виде **словаря** {ключ_словаря: соответствующее_значение}:

Ключ словаря: fields

Соответствующее значение: строка, которую вы написали бы в случае обычного sql запроса после ключевого слова SELECT

Ключ словаря: limit

Соответствующее значение: либо строка-пробел (' ') для выгрузки всей таблицы, либо строка типа LIMIT N, где N - количество необходимых строк

Все образцы ниже:

In [ ]:
```
engine = sqlalchemy.create_engine('postgresql://postgres:1234@localhost:5432
SCHEMA = 'bookings'
sql = {'bookings': None,
       'airports': None,
       'aircrafts': None,
       'flights': {'fields': "*, scheduled_departure + interval '1 day' as c
       'ticket_flights': {'fields': '*, md5((ticket_no || flight_id)::bytea)
                          'limit': ' '},
       'tickets': {'fields': 'ticket_no, book_ref, passenger_id, passenger_n
       'seats': {'fields': '*, md5((aircraft_code || seat_no)::bytea) AS key
```

Работа с CSV-файлами

Если исходные данные загружаются из csv-файлов, необходимо создать один основной словарь и один опциональный:

1. **Основной** (dates в образце) - конфиг для полей, содержащих даты для корректного парсинга. Формат - {table_name: [data_fields] }, где [data_fields] -

список из полей, которые содержат даты в пределах таблицы table_name

2. **Опциональный** (csv_names в образце) - для случая, когда имена csv файлов не совпадают с заданными ранее в словаре первичных ключей названиями таблиц (без учета расширения). Формат - {table_name: file_name.csv }

In [9]:
```
dates = {'bookings': ['book_date'],
         'flights': ['scheduled_departure',
                     'scheduled_arrival',
                     'actual_departure',
                     'actual_arrival',
                     'constraint_date']}
csv_names = {'tickets': 'tickets.csv',
             'bookings': 'bookings.csv',
             'flights': 'flights.csv',
             'ticket_flights': 'ticket_flights.csv',
             'airports': 'airports.csv',
             'seats': 'seats.csv',
             'aircrafts': 'aircrafts.csv'}
```

Загрузка данных

SQL interface

Необходимо передать sql-конфиг, схему в БД и sqlalchemy-движок.

В результате будет получен словарь, содержащий датафреймы. Каждый датафрейм - таблица из БД в соответствии с конфигом.

Документация функции и образец ниже:

In [10]:
```
help(mgf.read_sql)
```

```
Help on function read_sql in module multi_gen_func:

read_sql(config: 'dict', schema: 'str', engine: 'sqlalchemy.engine.base.Engin
e')
    Назначение:

    Последовательное чтение заранее определенной таблицы из нужной БД
    В теле функции для каждой таблицы и набора параметров вызывается функция
select_from_db

    Аргументы:

    config: словарь типа таблица: параметры, где параметры – None (арг по умо
лч.)
    или словарь, определяющий пользовательские значения аргумента/аргументов

    schema: схема в БД

    engine: sqlalchemy engine для нужной БД

    Результат:

    Выгруженная из БД схема связанных таблиц

    Возвращаемое значение:

    Словарь типа название_таблицы: соответствующий_pandas_dataframe
```

In [ ]:
```
df = mgf.read_sql(sql, SCHEMA, engine)
```

CSV interface.

Здесь по умолчанию необходимо передать путь к csv-файлам. Далее все зависит от того, совпадают ли имена таблиц и файлов (True по умолч):

1. Если да - второй аргумент True, передаем конфиг первичных ключей как аргумент pkeys
2. Если нет - False, далее нужно передать соответствующий имен csv файлов как аргумент names

В конце передается конфиг для полей с датой как аргумент dates, если таких полей нет, то его можно не передавать

В результате будет получен словарь, содержащий датафреймы. Каждый датафрейм - таблица из csv-файла в соответствии с конфигом

In [11]:
```
help(mgf.read_csv)
```

```
Help on function read_csv in module multi_gen_func:

read_csv(folder: 'str', same_names: 'bool', pkeys: 'dict' = {}, names: 'dict'
= {}, dates: 'dict' = {})
    Назначение:

    Последовательное чтение csv файлов в соответствии с аргументами. Если име
на
    таблиц и файлов совпадают (same_names=True), для определения таблиц
    используется аргумент pkeys. Если нет (same_names=False), то используется
    аргумент names.

    Аргументы:

    folder: папка с csv-файлами

    same_names: флаг совпадения имен файлов и имен таблиц

    pkeys: словарь типа имя_таблицы: первичный_ключ

    names: словарь типа имя_таблицы: имя_csv_файла

    dates: словарь типа имя_таблицы: поля, где поля - список полей, содержащи
х даты

    Результат:

    Прочитанная из csv-файлов схема связанных таблиц

    Возвращаемое значение:

    Словарь типа название_таблицы: соответствующий_pandas_dataframe
```

Имена совпадают:

In [12]:
```
df = mgf.read_csv('csvs/', True, pkeys=pkeys, dates=dates)
```

Имена не совпадают

In [ ]:
```
df = mgf.read_csv('csvs/', False, names=csv_names, dates=dates)
```

Предобработка данных

Связи между таблицами

Пояснение: sdv не может работать, если в связанных таблицах есть значения внешнего ключа, отсутствующие в значениях первичного ключа родительской таблице.

При полной выборке без лимитов можно отказаться от преобразований. Если же выборка неполная, то данной фукнции необходимо передать прежде полученный словарь **(далее схема)** из датафреймов и конфиг отношений и ключей

In [13]:
```
help(mgf.correct_relations)
```

```
Help on function correct_relations in module multi_gen_func:

correct_relations(df: 'dict', config: 'dict')
    Назначение:

    Коррекция отношений между таблицами для отсутствия соответствующих ошибок
в
    дальнейшем (устранение значений child_FK, которые не присутствуют в main_
PK)

    Аргументы:

    df: словарь типа название_таблицы: соответствующий_pandas_dataframe

    config: словарь типа таблицы: ключи, где
        таблицы - кортеж имен связанных таблиц, 0-ой элемент основная таблиц
а,
        1-ый - дочерняя. В случае наличия 2 и более связей между таблицами,
        необходимо добавить 2-ой элемент (или третий по счету) со случайными
        значением для обхода уникальности ключей словаря

        ключи - кортеж типа
        (поле_первичный_ключ_основной_таблицы, поле_внешний_ключ_дочерней_таб
лицы)

    Результат:

    Исправленные связи между таблицами

    Возвращаемое значение:

    Отсутствует
```

In [14]:
```
mgf.correct_relations(df, relations_and_keys)
```

Кастомная печать схемы:

In [15]:
```
mgf.print_len(df)
```

```
bookings 30000
tickets 1490
ticket_flights 2369
flights 30000
airports 104
aircrafts 9
seats 1339
```

Приведение полей с датами к оптимальному для работы типу

In [16]:
```
help(mgf.date_corrections_for_dataframe)
```

```
Help on function date_corrections_for_dataframe in module multi_gen_func:

date_corrections_for_dataframe(df: 'dict')
    Назначение:

    Приведение полей с датой каждой таблицы к оптимальному для дальнейших
    действий виду (удаление таймзоны)


    Аргументы:

    df: словарь из pandas dataframe для каждой таблицы

    Результат:

    Оптимальный формат дат во всех таблицах

    Возвращаемое значение:

    pandas dataframe
```

In [17]:
```
df = mgf.date_corrections_for_dataframe(df)
```

Метаданные. Основная конфигурация

В данном блоке:

1. Создается объект метаданных
2. Производится базовое автоматическое заполнение объекта на основе исходного DataFrame
3. Вызывается функция для основных дополнений объекта, подробное описание над соответствующей ячейкой

In [18]:
```
metadata = MultiTableMetadata()
```

In [19]:
```
mgf.detect_metadata(df, metadata)
```

Здесь производятся основные дополнения, а именно:

1. Ключевым полям присваивается тип id
2. Полям с датой присваивается тип datetime, а также формат даты
3. Определяются первичные ключи

Подробнее в документации ниже:

In [20]:
```
help(mgf.main_metadata_corrections)
```

```
Help on function main_metadata_corrections in module multi_gen_func:

main_metadata_corrections(df: 'dict', metadata: 'sdv.metadata.multi_table.Mul
tiTableMetadata', pkeys: 'dict', relations_and_keys: 'dict', regex: 'dict' =
{})
    Назначение:

    Выполнение основных корректировок объекта метадаты для всех заданных табл
иц,
    ключей и отношений
    В теле функции для каждой таблицы и соответствующих полей вызываются функ
ции:
    add_id_to_metadata - для первичных и внешних ключей
    date_correction_for_metadata - для полей, содержащих даты
    add_pk_to_metadata - для каждой таблицы

    Аргументы:

    df: словарь из датафреймов

    metadata: sdv multi_table_metadata

    pkeys: словарь типа имя_таблицы: первичный_ключ

    relations_and_keys: словарь типа таблицы: ключи, где

        таблицы - кортеж имен связанных таблиц, 0-ой элемент основная таблиц
а,
        1-ый - дочерняя. В случае наличия 2 и более связей между таблицами,
        необходимо добавить 2-ой элемент (или третий по счету) со случайными
        значением для обхода уникальности ключей словаря

        ключи - кортеж типа
        (поле_первичный_ключ_основной_таблицы, поле_внешний_ключ_дочерней_таб
лицы)

    regex: словарь типа имя_таблицы: поле_и_регулярка, где
        поле_и_регулярка - словарь типа имя_поля: регулярка

    Результат:

    Обновленный и полностью готовый к добавлению связей объект метаданных

    Возвращаемое значение:

    Отсутствует
```

In [21]:
```
mgf.main_metadata_corrections(df, metadata, pkeys, relations_and_keys)
```

Базовый объект метаданных выглядит следующим

образом:

In [22]:
```
metadata
```

Out[22]:

```
{
    "tables": {
        "bookings": {
            "columns": {
                "book_ref": {
                    "sdtype": "id"
                },
                "book_date": {
                    "sdtype": "datetime",
                    "datetime_format": "%Y-%m-%d %H:%M:%S"
                },
                "total_amount": {
                    "sdtype": "numerical"
                }
            },
            "primary_key": "book_ref"
        },
        "tickets": {
            "columns": {
                "ticket_no": {
                    "sdtype": "id"
                },
                "book_ref": {
                    "sdtype": "id"
                },
                "passenger_id": {
                    "sdtype": "categorical"
                },
                "passenger_name": {
                    "sdtype": "categorical"
                }
            },
            "primary_key": "ticket_no"
        },
        "ticket_flights": {
            "columns": {
                "ticket_no": {
                    "sdtype": "id"
                },
                "flight_id": {
                    "sdtype": "id"
                },
                "fare_conditions": {
                    "sdtype": "categorical"
                },
                "amount": {
                    "sdtype": "numerical"
                },
                "key": {
                    "sdtype": "id"
                }
            },
            "primary_key": "key"
        },
        "flights": {
            "columns": {
```

```
            "flight_id": {
                "sdtype": "id"
            },
            "flight_no": {
                "sdtype": "categorical"
            },
            "scheduled_departure": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "scheduled_arrival": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "departure_airport": {
                "sdtype": "id"
            },
            "arrival_airport": {
                "sdtype": "id"
            },
            "status": {
                "sdtype": "categorical"
            },
            "aircraft_code": {
                "sdtype": "id"
            },
            "actual_departure": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "actual_arrival": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "constraint_date": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            }
        },
        "primary_key": "flight_id"
    },
    "airports": {
        "columns": {
            "airport_code": {
                "sdtype": "id"
            },
            "airport_name": {
                "sdtype": "categorical"
            },
            "city": {
                "sdtype": "categorical"
            },
            "coordinates": {
                "sdtype": "categorical"
            },
            "timezone": {
```

```
                    "sdtype": "categorical"
                }
            },
            "primary_key": "airport_code"
        },
        "aircrafts": {
            "columns": {
                "aircraft_code": {
                    "sdtype": "id"
                },
                "model": {
                    "sdtype": "categorical"
                },
                "range": {
                    "sdtype": "numerical"
                }
            },
            "primary_key": "aircraft_code"
        },
        "seats": {
            "columns": {
                "aircraft_code": {
                    "sdtype": "id"
                },
                "seat_no": {
                    "sdtype": "categorical"
                },
                "fare_conditions": {
                    "sdtype": "categorical"
                },
                "key": {
                    "sdtype": "id"
                }
            },
            "primary_key": "key"
        }
    },
    "relationships": [],
    "METADATA_SPEC_VERSION": "MULTI_TABLE_V1"
}
```

Опциональные дополнения

Обезличить неключевые поля:

In [23]:
```
help(mgf.hide_not_key_fields)
```

```
Help on function hide_not_key_fields in module multi_gen_func:

hide_not_key_fields(metadata: 'sdv.metadata.multi_table.MultiTableMetadata',
other_important_fields: 'dict', regex: 'dict' = {})
    Назначение:

    Добавление в объект метадаты определения для оставшихся выбранных полей,
    выбранных таблиц, которые должны иметь тип id

    В теле функции вызывается для определенных таблицы и полей
    вызывается add_id_to_metadata
    Аргументы:

    meta: sdv multi_table_metadata

    other_important_fields: словарь типа имя_таблицы: поле_таблицы

    regex: словарь типа имя_таблицы: поле_и_регулярка, где
        поле_и_регулярка - словарь типа имя_поля: регулярка

    Результат:

    Обновленный объект метаданных - дополнительные неключевые поля типа id

    Возвращаемое значение:

    Отсутствует
```

In [24]:
```
mgf.hide_not_key_fields(metadata, other_important_fields)#, regex)
```

Добавить кастомные типы:

In [25]:
```
help(mgf.add_custom_type)
```

```
Help on function add_custom_type in module multi_gen_func:

add_custom_type(metadata: 'sdv.metadata.multi_table.MultiTableMetadata', othe
r_types_fields: 'dict')
    Назначение:

    Добавление в объект метадаты определения для выбранных полей выбранных та
блиц,
    которым хотим сопоставить кастомный тип (из sdv или Faker)

    Аргументы:

    metadata: sdv multi_table_metadata

    other_types_fields: словарь типа имя_таблицы: поле_и_тип, где
        поле_и_тип - список типа [имя_поля, тип]

    Результат:

    Обновленный объект метаданных - кастомные типы для всех определенных поле
й

    Возвращаемое значение:

    Отсутствует
```

In [26]:
```
mgf.add_custom_type(metadata, other_types_fields)
```

Добавление связей между таблицами

In [27]:
```
help(mgf.add_relations)
```

```
Help on function add_relations in module multi_gen_func:

add_relations(metadata: 'sdv.metadata.multi_table.MultiTableMetadata', relati
ons_and_keys: 'dict')
    Назначение:
    Добавление в объект метадаты всех определенных отношений между таблицами

    Аргументы:

    metadata: sdv multi_table_metadata

    relations_and_keys: словарь типа таблицы: ключи, где

        таблицы – кортеж имен связанных таблиц, 0-ой элемент основная таблиц
а,
        1-ый – дочерняя. В случае наличия 2 и более связей между таблицами,
        необходимо добавить 2-ой элемент (или третий по счету) со случайными
        значением для обхода уникальности ключей словаря

        ключи – кортеж типа
        (поле_первичный_ключ_основной_таблицы, поле_внешний_ключ_дочерней_таб
лицы)

    Результат:

    Обновленный объект метаданных – все определенные в конфиге отношения

    Возвращаемое значение:

    Отсутствует
```

In [28]:
```
mgf.add_relations(metadata, relations_and_keys)
```

Итоговая метадата

In [29]:
```
metadata
```

Out[29]:

```
{
    "tables": {
        "bookings": {
            "columns": {
                "book_ref": {
                    "sdtype": "id"
                },
                "book_date": {
                    "sdtype": "datetime",
                    "datetime_format": "%Y-%m-%d %H:%M:%S"
                },
                "total_amount": {
                    "sdtype": "numerical"
                }
            },
            "primary_key": "book_ref"
        },
        "tickets": {
            "columns": {
                "ticket_no": {
                    "sdtype": "id"
                },
                "book_ref": {
                    "sdtype": "id"
                },
                "passenger_id": {
                    "sdtype": "id"
                },
                "passenger_name": {
                    "sdtype": "name"
                }
            },
            "primary_key": "ticket_no"
        },
        "ticket_flights": {
            "columns": {
                "ticket_no": {
                    "sdtype": "id"
                },
                "flight_id": {
                    "sdtype": "id"
                },
                "fare_conditions": {
                    "sdtype": "categorical"
                },
                "amount": {
                    "sdtype": "numerical"
                },
                "key": {
                    "sdtype": "id"
                }
            },
            "primary_key": "key"
        },
        "flights": {
            "columns": {
```

```
            "flight_id": {
                "sdtype": "id"
            },
            "flight_no": {
                "sdtype": "id"
            },
            "scheduled_departure": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "scheduled_arrival": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "departure_airport": {
                "sdtype": "id"
            },
            "arrival_airport": {
                "sdtype": "id"
            },
            "status": {
                "sdtype": "categorical"
            },
            "aircraft_code": {
                "sdtype": "id"
            },
            "actual_departure": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "actual_arrival": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            },
            "constraint_date": {
                "sdtype": "datetime",
                "datetime_format": "%Y-%m-%d %H:%M:%S"
            }
        },
        "primary_key": "flight_id"
    },
    "airports": {
        "columns": {
            "airport_code": {
                "sdtype": "id"
            },
            "airport_name": {
                "sdtype": "categorical"
            },
            "city": {
                "sdtype": "categorical"
            },
            "coordinates": {
                "sdtype": "categorical"
            },
            "timezone": {
```

```
                "sdtype": "categorical"
            }
        },
        "primary_key": "airport_code"
    },
    "aircrafts": {
        "columns": {
            "aircraft_code": {
                "sdtype": "id"
            },
            "model": {
                "sdtype": "categorical"
            },
            "range": {
                "sdtype": "numerical"
            }
        },
        "primary_key": "aircraft_code"
    },
    "seats": {
        "columns": {
            "aircraft_code": {
                "sdtype": "id"
            },
            "seat_no": {
                "sdtype": "id"
            },
            "fare_conditions": {
                "sdtype": "categorical"
            },
            "key": {
                "sdtype": "id"
            }
        },
        "primary_key": "key"
    }
},
"relationships": [
    {
        "parent_table_name": "bookings",
        "child_table_name": "tickets",
        "parent_primary_key": "book_ref",
        "child_foreign_key": "book_ref"
    },
    {
        "parent_table_name": "tickets",
        "child_table_name": "ticket_flights",
        "parent_primary_key": "ticket_no",
        "child_foreign_key": "ticket_no"
    },
    {
        "parent_table_name": "flights",
        "child_table_name": "ticket_flights",
        "parent_primary_key": "flight_id",
        "child_foreign_key": "flight_id"
    },
```

```
            {
                "parent_table_name": "airports",
                "child_table_name": "flights",
                "parent_primary_key": "airport_code",
                "child_foreign_key": "departure_airport"
            },
            {
                "parent_table_name": "airports",
                "child_table_name": "flights",
                "parent_primary_key": "airport_code",
                "child_foreign_key": "arrival_airport"
            },
            {
                "parent_table_name": "aircrafts",
                "child_table_name": "flights",
                "parent_primary_key": "aircraft_code",
                "child_foreign_key": "aircraft_code"
            },
            {
                "parent_table_name": "aircrafts",
                "child_table_name": "seats",
                "parent_primary_key": "aircraft_code",
                "child_foreign_key": "aircraft_code"
            }
        ],
        "METADATA_SPEC_VERSION": "MULTI_TABLE_V1"
    }
```

Можно сохранить в словарь или в json:

In [30]:
```
help(mgf.save_metadata)
```

```
Help on function save_metadata in module multi_gen_func:

save_metadata(metadata: 'sdv.metadata.multi_table.MultiTableMetadata', mode:
'str', name: 'str' = None)
    Назначение:

    Сохранение объекта метадаты в словарь или json

    Аргументы:

    metadata: sdv multi_table_metadata

    mode: тип сохранения – 'dict' или 'json'

    name: None по умолч, путь к json файлу в случе mode='json'

    Результат:

    Сохраненные в словарь метаданные

    Возвращаемое значение:

    Словарь с метадатой или сохраненный json-файл
```

In [ ]:
```
meta_dict = mgf.save_metadata(metadata, 'dict')
```

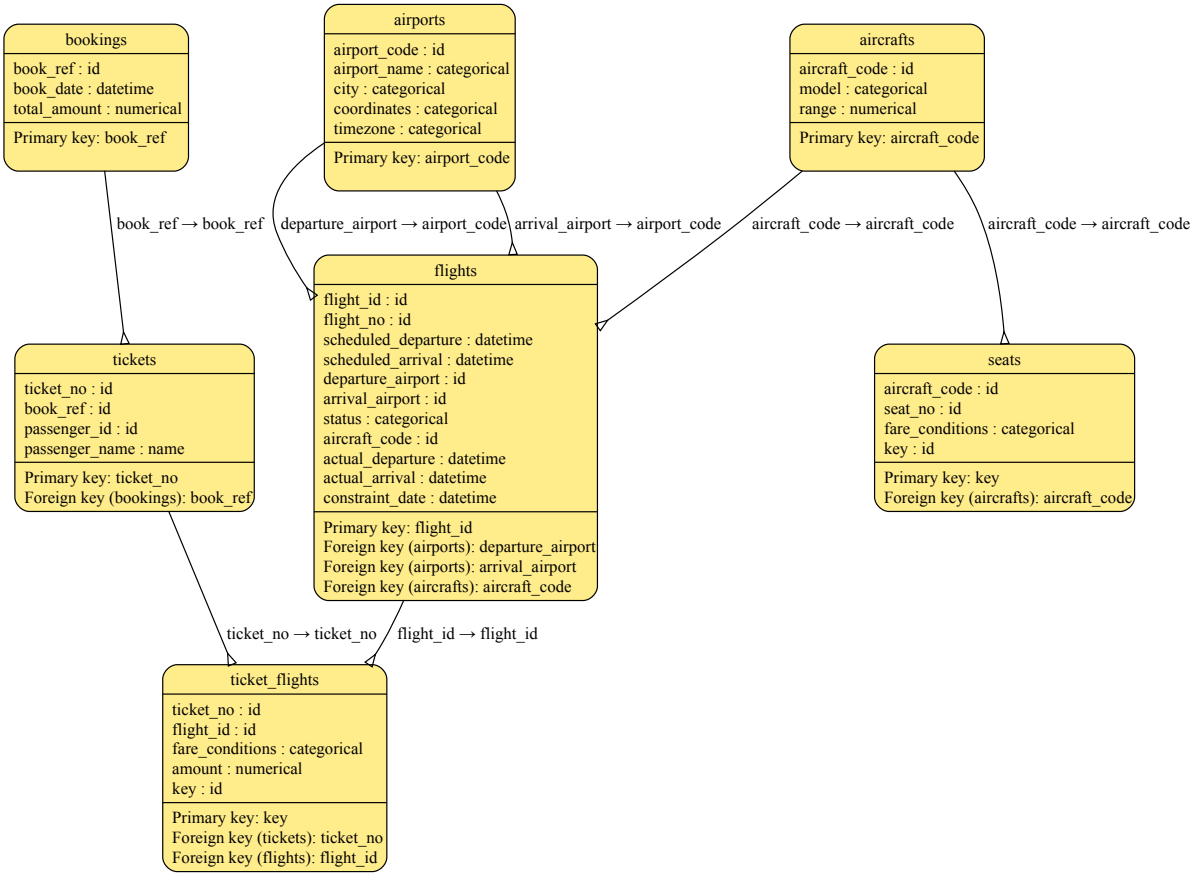
Валидация позволит проверить наши метаданные на соответствие логике пакета sdv

In [31]:
```
metadata.validate()
```

ERD на основе созданных метаданных. Можно проверить и сравнить с исходной

In [32]:
```
metadata.visualize()
```

Out[32]:



Генерация

Для генерации связанных таблиц имеется только одна open-source модель, а именно HMA:

https://docs.sdv.dev/sdv/multi-table-data/modeling/synthesizers/hmasynthesizer

Создание модели

Передается метадата, также можно передать параметр локализации (с ним, например, имена будут генерироваться в соответствии с регионом).

Ограничение: локализация работает с переменным успехом

In [33]:
```
multi_synthesizer = HMASynthesizer(metadata, locales=['ru_RU'])
```

Добавление ограничений/условий

Здесь можно добавить ограничения на столбцы таблиц. Функция позволяет легко добавить 2 ограничения - Inequality и FixedCombinations. Подробнее о них, а также о других доступных:

https://docs.sdv.dev/sdv/multi-table-data/modeling/synthetic-data-workflow/constraints

In [34]:
```
help(mgf.add_constraint)
```

```
Help on function add_constraint in module multi_gen_func:

add_constraint(model: 'sdv.multi_table.hma.HMASynthesizer', constraint_class:
'str', table: 'str', parameters: 'list')
    Назначение:

    Добавление ограничений к объекту synthesizer

    Аргументы:

    model: объекту sdv synthesizer

    constraint_class: класс ограничения - 'Inequality' 'FixedCombinations'

    table: имя таблицы, к которой применяются ограничения

    parameters: список параметров, соответствующих типу ограничения

    Результат:

    Модель дополнена ограничением

    Возвращаемое значение:

    Отсутствует
```

In [35]:
```
mgf.add_constraint(multi_synthesizer, 'inequality', 'flights', ['scheduled_d
```

In [36]:
```
mgf.add_constraint(multi_synthesizer, 'inequality', 'flights', ['scheduled_a
```

Валидация позволит проверить соответствие исходных данных с заполненными ранее метаданными

In [37]:
```
multi_synthesizer.validate(df)
```

Обучение модели

In [ ]:
```
multi_synthesizer.fit(df)
```

Выборка синтетических данных

Параметром scale передается % от размера исходных данных. Пропорция не всегда будет точной, все зависит от связей между таблицами

In [ ]:
```
synthetic_schema = multi_synthesizer.sample(scale=0.1)
```

Сравнение исходных и синтетических данных

In [40]:
```
df['bookings']
```

Out[40]:

	book_ref	book_date	total_amount
0	00000F	2017-07-05 00:12:00	265700.0
1	000012	2017-07-14 06:02:00	37900.0
2	00002D	2017-05-20 15:45:00	114700.0
3	000068	2017-08-15 11:27:00	18100.0
4	0000C9	2017-06-30 12:52:00	54600.0
...	...	...	...
29995	0CF4F4	2017-05-06 15:45:00	57500.0
29996	0CF513	2017-06-01 08:25:00	71100.0
29997	0CF533	2017-05-26 20:07:00	29100.0
29998	0CF546	2017-05-26 18:37:00	86300.0
29999	0CF561	2017-08-09 18:27:00	47700.0

30000 rows × 3 columns

In [41]:
```
synthetic_schema['bookings']
```

Out[41]:

	book_ref	book_date	total_amount
0	00000	2017-07-02 01:44:08	57980.0
1	00001	2017-07-01 21:26:22	53709.0
2	0000a	2017-04-29 06:11:26	18382.0
3	0000b	2017-06-05 17:29:22	27236.0
4	0000c	2017-07-03 07:39:51	46960.0
...	...	...	...
2995	00buz	2017-07-20 10:41:26	31599.0
2996	00bv0	2017-07-01 14:39:53	31974.0
2997	00bv1	2017-07-06 21:10:37	69102.0
2998	00bva	2017-05-25 09:33:13	164950.0
2999	00bvb	2017-07-13 16:44:12	81039.0

3000 rows × 3 columns

In [42]:
```
df['tickets']
```

Out[42]:

	ticket_no	book_ref	passenger_id	passenger_name
0	5435610134	03A60B	2684 451872	NATALYA IVANOVA
1	5435610186	0814EC	0453 836625	SERGEY KUDRYASHOV
2	5435610187	0814EC	3138 792180	LYUBOV ARKHIPOVA
3	5435610188	0814EC	8156 424995	LYUBOV ANDREEVA
4	5435610189	0BC514	3753 801865	VLADIMIR NAZAROV
...	...	...	...	...
1485	5435731551	09DBD6	9592 072127	LARISA KOLESNIKOVA
1486	5435731568	014392	2026 301418	MARIYA CHERNOVA
1487	5435731640	01459E	0174 867003	ALEKSANDR SOROKIN
1488	5435731705	09BD24	5444 971078	LYUBOV KUZNECOVA
1489	5435731713	0130D1	2641 670309	IGOR DENISOV

1490 rows × 4 columns

In [43]:
```
synthetic_schema['tickets']
```

Out[43]:

	ticket_no	book_ref	passenger_id	passenger_name
0	0	00000	00000	Jesus Williams
1	1	00001	00001	Hannah Sanders
2	2	0000a	0000a	Sabrina Carr
3	3	0000b	0000b	Melanie Paul
4	4	0000c	0000c	Danielle Murphy
...	...	...	...	...
2995	2995	00buz	00buz	Michael Smith
2996	2996	00bv0	00bv0	Mary Nguyen
2997	2997	00bv1	00bv1	Melissa Hancock
2998	2998	00bva	00bva	Leslie Weber
2999	2999	00bvb	00bvb	Jeffrey Phillips

3000 rows × 4 columns

In [44]:
```
df['ticket_flights']
```

Out[44]:

	ticket_no	flight_id	fare_conditions	amount	
0	5435723005	12715	Economy	12200.0	64c653e96297e54daf6f86e65be
1	5435723045	12643	Economy	12200.0	274d064dc5f0598c3724efba1d8
2	5435722914	12658	Economy	12200.0	f2c03804e7957134097368ccbfl
3	5435722906	12691	Economy	12200.0	38031346148cdd74eee8f14c86
4	5435723009	12622	Economy	12200.0	2fc2e106179ca6b2a84374b63c0
...	...	...	...	...	
2364	5435679681	13409	Economy	17000.0	06ca55efccdaae98e62ceff43d9
2365	5435714978	14073	Economy	29000.0	d7dc45176fc97abbe99a8bf5c7
2366	5435651611	3260	Economy	16700.0	950fcf6eb50297bb4c4f0a8213
2367	5435626121	21810	Economy	3200.0	325f72251557878116423b9dd3a
2368	5435681071	13368	Economy	17000.0	d27671fb693b90fd322f4858a19

2369 rows × 5 columns

In [45]:
```
synthetic_schema['ticket_flights']
```

Out[45]:

	ticket_no	flight_id	fare_conditions	amount	key
0	0	8991	Business	23550.0	00000
1	0	102	Business	22754.0	00001
2	1	3558	Comfort	37393.0	0000a
3	1	8411	Comfort	33010.0	0000b
4	2	10639	Comfort	32971.0	0000c
...	...	...	...	...	...
6043	2997	1418	Comfort	35607.0	00frv
6044	2998	172	Comfort	31142.0	00frw
6045	2998	11691	Comfort	29379.0	00frx
6046	2999	780	Economy	18791.0	00fry
6047	2999	282	Business	19717.0	00frz

6048 rows × 5 columns

In [46]:
```
df['flights']
```

Out[46]:

	flight_id	flight_no	scheduled_departure	scheduled_arrival	departure_airpor
0	182	PG0402	2017-09-01 09:25:00	2017-09-01 10:20:00	DME
1	1996	PG0335	2017-08-26 06:30:00	2017-08-26 08:35:00	DME
2	5979	PG0384	2017-08-26 09:10:00	2017-08-26 09:40:00	DME
3	8136	PG0138	2017-08-28 07:15:00	2017-08-28 08:20:00	VKO
4	10455	PG0277	2017-09-12 08:45:00	2017-09-12 12:10:00	SVO
...	...	...	...	...	..
29995	29984	PG0493	2017-07-27 13:50:00	2017-07-27 17:25:00	PEE
29996	29985	PG0493	2017-07-23 13:50:00	2017-07-23 17:25:00	PEE
29997	29986	PG0492	2017-05-31 08:20:00	2017-05-31 11:55:00	PEE
29998	29987	PG0492	2017-07-23 08:20:00	2017-07-23 11:55:00	PEE
29999	29988	PG0492	2017-05-26 08:20:00	2017-05-26 11:55:00	PEE

30000 rows × 11 columns

In [47]:
```
synthetic_schema['flights']
```

Out[47]:

	flight_id	flight_no	scheduled_departure	scheduled_arrival	departure_airport
0	0	00000	2017-11-19 14:56:37	2017-11-19 23:46:37	00000
1	1	00001	2017-10-04 22:16:27	2017-10-05 07:06:27	00000
2	2	0000a	2017-09-06 07:18:51	2017-09-06 16:08:51	00000
3	3	0000b	2017-11-21 07:36:36	2017-11-21 16:26:36	00000
4	4	0000c	2017-08-25 15:42:23	2017-08-26 00:32:23	00000
...	...	...	...	...	..
12602	12602	00oaa	2017-09-20 20:31:21	2017-09-20 22:40:01	0000g
12603	12603	00oab	2017-08-23 00:53:09	2017-08-23 02:59:09	0000g
12604	12604	00oac	2017-07-04 17:15:27	2017-07-04 19:00:23	0000g
12605	12605	00oad	2017-10-07 05:42:47	2017-10-07 07:41:28	0000g
12606	12606	00oae	2017-11-01 02:48:12	2017-11-01 06:24:17	0000h

12607 rows × 11 columns

In [48]:
```
df['airports']
```

Out[48]:

	airport_code	airport_name	city	coor
0	YKS	Якутск	Якутск	(129.77099609375,62.0932998657
1	MJZ	Мирный	Мирный	(114.03900146484375,62.534698486
2	KHV	Хабаровск-Новый	Хабаровск	(135.18800354004,48.527999
3	PKC	Елизово	Петропавловск-Камчатский	(158.45399475097656,53.167900085
4	UUS	Хомутово	Южно-Сахалинск	(142.71800231933594,46.888698577
...	...	...	...	
99	MMK	Мурманск	Мурманск	(32.75080108642578,68.781700134
100	ABA	Абакан	Абакан	(91.38500213623047,53.74000167
101	BAX	Барнаул	Барнаул	(83.53849792480469,53.363800048
102	AAQ	Витязево	Анапа	(37.347301483154,45.0021018
103	CNN	Чульман	Нерюнгри	(124.91400146484,56.9138984

104 rows × 5 columns

In [49]:
```
synthetic_schema['airports']
```

Out[49]:

	airport_code	airport_name	city	coordinates
0	00000	Ижевск	Советский	(53.45750045776367,56.82809829711914)
1	00001	Братск	Братск	(101.697998046875,56.370601654052734)
2	0000a	Когалым	Псков	(92.493301391602,56.172901153564)
3	0000b	Байкал	Мурманск	(107.43800354003906,51.80780029296875)
4	0000c	Пулково	Санкт-Петербург	(20.592599868774414,54.88999938964844)
5	0000d	Саратов-Центральный	Ноябрьск	(46.04669952392578,51.564998626708984)
6	0000e	Сыктывкар	Тюмень	(50.16429901123,53.504901885986)
7	0000f	Советский	Ижевск	(53.45750045776367,56.82809829711914)
8	0000g	Донское	Усть-Кут	(41.482799530029,52.806098937988)
9	0000h	Братск	Братск	(101.697998046875,56.370601654052734)

In [50]:
```
df['aircrafts']
```

Out[50]:

	aircraft_code	model	range
0	773	Боинг 777-300	11100
1	763	Боинг 767-300	7900
2	SU9	Сухой Суперджет-100	3000
3	320	Аэробус A320-200	5700
4	321	Аэробус A321-200	5600
5	319	Аэробус A319-100	6700
6	733	Боинг 737-300	4200
7	CN1	Сессна 208 Караван	1200
8	CR2	Бомбардье CRJ-200	2700

In [51]:
```
synthetic_schema['aircrafts']
```

Out[51]:

	aircraft_code	model	range
0	00000	Бомбардье CRJ-200	2921
1	00001	Аэробус A319-100	3505
2	0000a	Сухой Суперджет-100	6718
3	0000b	Аэробус A319-100	5562
4	0000c	Бомбардье CRJ-200	2811
5	0000d	Сессна 208 Караван	3281
6	0000e	Боинг 737-300	3818
7	0000f	Боинг 777-300	9409
8	0000g	Боинг 737-300	5573

In [52]:
```
df['seats']
```

Out[52]:

	aircraft_code	seat_no	fare_conditions	key
0	319	2A	Business	24a1c4748bdb1595652850e5f50e36eb
1	319	2C	Business	c900df0c56ce4b6fedb460de266879fd
2	319	2D	Business	c8b6370a08bcecc904de84dcac244522
3	319	2F	Business	f4289948b729ee3e12dd1397d4785a6d
4	319	3A	Business	02daa5e038a32344238311b846a277c5
...	...	...	...	...
1334	773	48H	Economy	7bc0d45df3958223872ca14f0ded068f
1335	773	48K	Economy	2e571152a79aa9144a20e5de9d7c5f03
1336	773	49A	Economy	e6d2a03c4d4795077449521c4e874819
1337	773	49C	Economy	0c1d66c7f15702328123e8163afc811e
1338	773	49D	Economy	bf5a7c61ef659ea05ee0c02b363ca0d2

1339 rows × 4 columns

In [53]:
```
synthetic_schema['seats']
```

Out[53]:

	aircraft_code	seat_no	fare_conditions	key
0	00000	00000	Economy	00000
1	00000	00001	Economy	00001
2	00000	0000a	Economy	0000a
3	00000	0000b	Economy	0000b
4	00000	0000c	Economy	0000c
...	...	...	...	...
1221	0000g	001np	Comfort	001np
1222	0000g	001nq	Comfort	001nq
1223	0000g	001nr	Comfort	001nr
1224	0000g	001ns	Comfort	001ns
1225	0000g	001nt	Comfort	001nt

1226 rows × 4 columns

Отчеты о качестве

Создание отчета

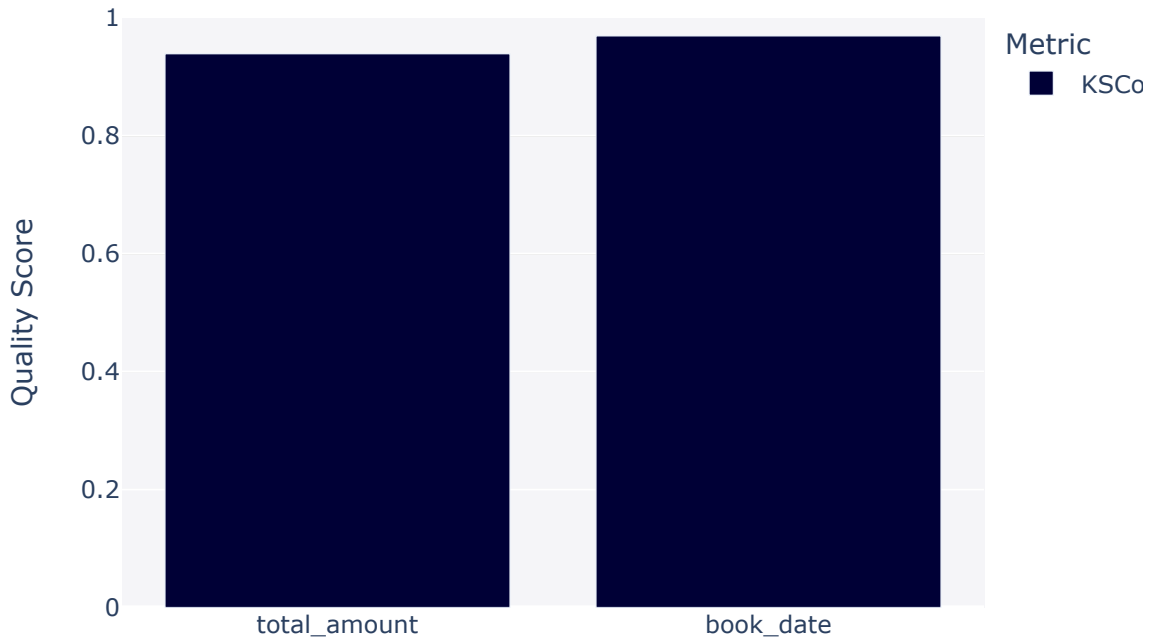
Это основной способ автоматической оценки качества синтетических данных. В случае моделирования связанных таблиц оценка качества не совсем честная, т.к. синтетическая выборка пропорциями так или иначе будет отличаться от исходной (аргумент scale < 1 в блоке выборки). Кроме этого, если не задать шаблоны регулярок, то поля типа id будут почти всегда отличаться по формату, а значит и по значениям. Поэтому пугаться полученных 40-60% не стоит. Главное, что данные обезличены, а связи сохранены

In [54]:
```
quality_report = evaluate_quality(
    df,
    synthetic_schema,
    metadata
)
```

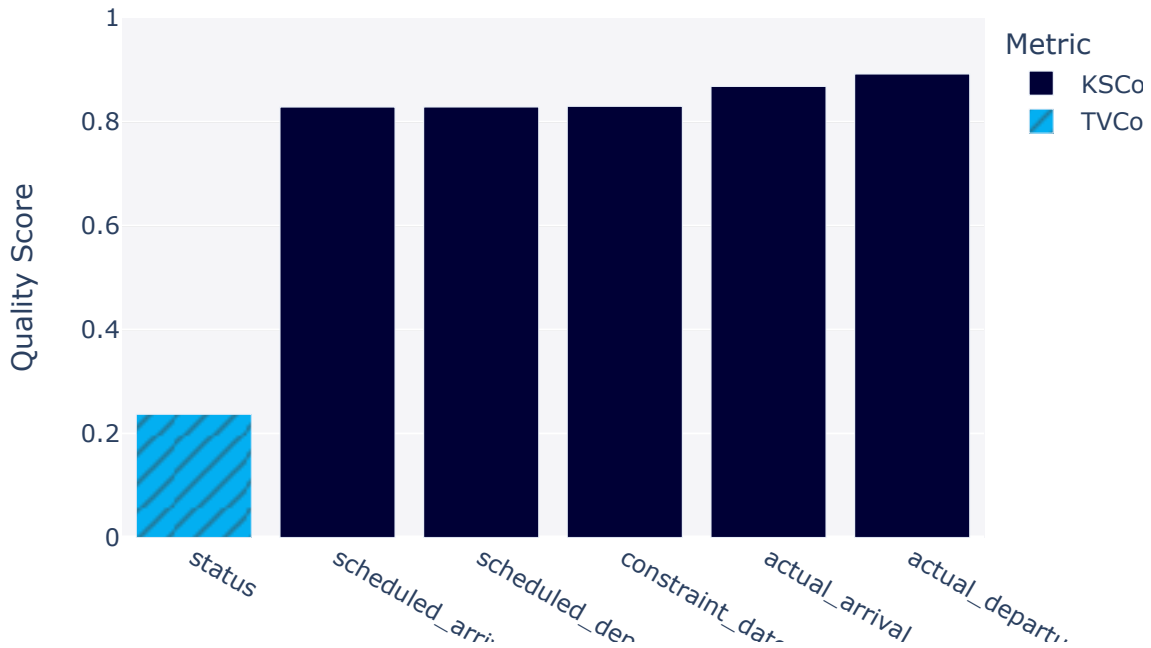
```
Creating report: 100%|██████████| 5/5 [00:00<00:00,  5.29it/s]
Overall Quality Score: 47.73%

Properties:
Column Shapes: 54.33%
Column Pair Trends: 45.67%
Parent Child Relationships: 43.2%
```

Визуальное сравнение распределения и границ полей (категориальных, числовых и полей с датами) таблиц в исходных и синтетических данных

In [55]:
```
fig = quality_report.get_visualization('Column Shapes', table_name='bookings
fig.show()
```



In [56]:
```
fig = quality_report.get_visualization('Column Shapes', table_name='flights'
fig.show()
```

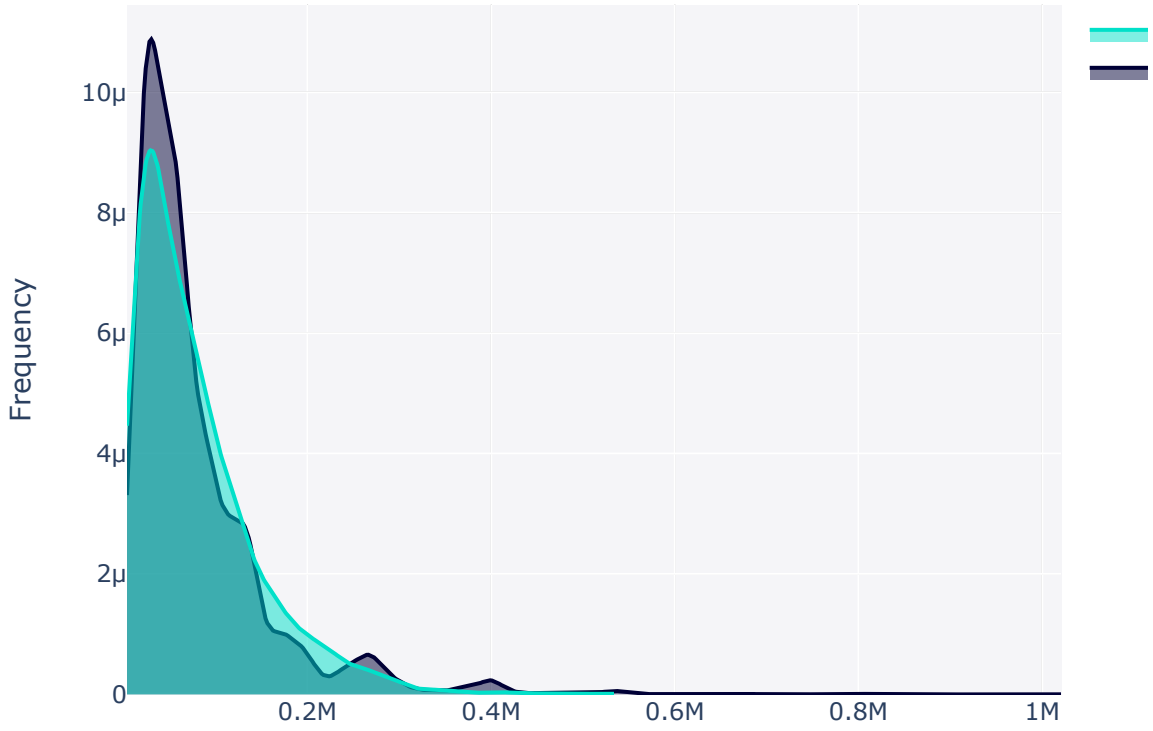


Визуальная проверка распределения конкретного поля отдельной таблицы (категориального, числового, поля-даты или bool)

In [57]:
```
help(mgf.plot_field_distribution)
```

```
Help on function plot_field_distribution in module multi_gen_func:

plot_field_distribution(real_data: 'dict', gen_data: 'dict', metadata: 'sdv.m
ulti_table.hma.HMASynthesizer', table: 'str', field: 'str')
    Назначение:

    Создание графика, отражающего распределение переменной с типом
    numerical, categorical, datetime, bool

    Аргументы:

    real_data: исходные данные – словарь из датафреймов

    gen_data: синтетические данные – словарь из датафреймов

    metadata: MultiTableMetadata

    table: имя таблицы

    field: имя поля (переменной)

    Результат:

    Построенный интерактивный график распределения переменной

    Возвращаемое значение:

    Отсутствует
```

In [58]:
```
mgf.plot_field_distribution(df, synthetic_schema, metadata, 'bookings', 'tot
```



Отчет-диагностика

Здесь можно получить информацию о соблюдении границ значений, доле отсутствующих промежутков или категорий,а также копий исходных данных

Пояснение про копии исходных данных: Если не добавлять шаблоны регулярок, то к данным копиям относятся либо категориальные данные (которые было решено не обезличивать), либо даты/числа/bool.

Как и ранее, если совпало значение в поле с датой, а остальные реально важные поля обезличены, то такое совпадение включится в отчет, хотя по факту оно безобидно и переживать не стоит

In [59]:
```
diagnostic_report = run_diagnostic(
    real_data=df,
    synthetic_data=synthetic_schema,
    metadata=metadata)
```

```
Creating report: 100%|██████████| 4/4 [00:51<00:00, 12.89s/it]
```

```
DiagnosticResults:

SUCCESS:
✓ The synthetic data follows over 90% of the min/max boundaries set by the re
al data

WARNING:
! The synthetic data is missing more than 10% of the numerical ranges present
in the real data
! The synthetic data is missing more than 10% of the categories present in th
e real data
! More than 10% of the synthetic rows are copies of the real data
```