# ПРИЛОЖЕНИЕ А Текст программной системы

ВВЕДЕНИЕ

Данный программный продукт предназначен для автоматизации процесса проверки валидности наборов выгрузок компаний партнеров компании «Ланит Омни». Данные выгрузки являются отчётными данными по программе лояльности, которую внедряет компания «Ланит Омни». На основе присылаемых данных далее проводится предиктивная аналитика, с помощью которой аналитики компании показывают, как можно повысить средний чек, количество покупателей в магазине-партнёре. Единственной и самой важной задачей является сделать стабильное приложение, которое поможет отделу аналитики быстрее составлять отчёты о найденных ошибках при заполнении БД компании партнёрами.

Для разработки был выбран язык программирования Python 3.6 с технологией анализа данных pandas.

Для создания данного программного продукта была выбрана инструментальная среда JetBrains PyCharm 2018.1.1.

Стандартные библиотеки:

Таблица 1 - библиотеки

|  |  |
| --- | --- |
| № | Название библиотеки |
| 1 | Pandas |
| 2 | Itertools |
| 3 | Collections |
| 4 | Datetime |
| 5 | Os |
| 6 | \_\_future\_\_ |
| 7 | Json |
| 8 | Sys |

Разработанные модули:

Таблица 2 - модули

| № | Название модуля |
| --- | --- |
| 1 | Run.py |
| 2 | Main.py |
| 3 | Validator.py |
| 4 | Statistic.py |
| 5 | XML\_to\_DF.py |
| 6 | catalogConfigs.json |
| 7 | clientsConfig.json |
| 8 | itemsConfig.json |
| 9 | saleConfig.json |
| 10 | Shop\_test\_config.json |
| 11 | shopsConfig.json |
| 12 | statusConfig.json |
| Итого | 12 модулей |

Перечень модулей

Список авторских и смешанных модулей:

Таблица 3 - перечень модулей

| № | Название модуля | Выполняемые действия | Размер в строках |
| --- | --- | --- | --- |
| 1 | Run.py | Содержит вызов всех методов приложения | 38 |
| 2 | Main.py | Содержит в себе код для проверки всех файлов, указанных в требованиях к выгрузкам | 19 |
| 3 | Validator.py | Содержит в себе код для проверки валидности данных | 348 |
| 4 | Statistic.py | Содержит в себе код для сбора статистики данных | 59 |
| 5 | XML\_to\_DF.py | Содержит в себе код для преобразования xml-структуры в структуру DataFrame | 11 |
| 6 | catalogConfigs.json | Файл с параметрами проверки | 40 |
| 7 | clientsConfig.json | Файл с параметрами проверки | 85 |
| 8 | itemsConfig.json | Файл с параметрами проверки | 58 |
| 9 | saleConfig.json | Файл с параметрами проверки | 137 |
| 10 | Shop\_test\_config.json | Файл с параметрами проверки | 22 |
| 11 | shopsConfig.json | Файл с параметрами проверки | 53 |
| 12 | statusConfig.json | Файл с параметрами проверки | 44 |
| Итого: | | | 914 |

Список заимствованных модулей (библиотек):

Таблица 4 - библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| № | Название библиотеки | Назначение | Версия |
| 1 | Pandas | Анализ данных | 0.23.0 |
| 2 | Itertools | Создание итераторов | Входят в стандартный набор библиотек conda 4.5.4 |
| 3 | Collections | Подсчёт кол-ва значений |
| 4 | Datetime | Работа с датой и временем |
| 5 | Os | Работа с операционной системой |
| 6 | \_\_future\_\_ | Смена порядка анализа модуля |
| 7 | Json | Работа с файлами с параметрами |
| 8 | Sys | Список аргументов командной строки |

Модуль Main.py:

from Run import Run

def Test():

x = ''

while x != 'n':

print("Print 'y' to begin to validate or 'n' to end process")

x = input()

if x == "y":

print("Print path to file, which you want to validate")

Run(input())

elif x == "n":

print("--------------END--------------")

else:

print("You made mistake. Choose 'y' for continue validate or 'n' to end process")

print(Test())

Модуль XML\_to\_DF.py:

from xmlutils.xml2csv import xml2csv

# метод преобразования xml файла в формат DataFrame

def convert(input\_file, output\_file):

converter = xml2csv(input\_file, output\_file)

converter.convert(tag="Каталог")

convert(input(), input())

Модуль Statistic.py

# Импорт библиотек

from \_\_future\_\_ import print\_function

import pandas as pd

from collections import Counter

class Statistic:

# Метод подсчёта строк в столбце

def CountLenOfRow(self, df, column):

if df[column].dtype in ('object', 'string\_', 'unicode\_'):

print(" Name column {}".format(column))

print(" Max length of cell {}".format(df[column].map(len).max()))

# Метод нахождения минимального и максимального значений в столбцах

def CountMaxMin(self, df, column):

df[column] = pd.to\_numeric(df[column], errors='coerce')

print(" Name column {}".format(column))

print(" Max value {}".format(df[column].max()))

print(" Min value {}".format(df[column].max()))

print(" Min value {}".format(df[column].min()))

# Метод инициализирующий все методы сбора статистики

def DFStatistic(self, df, column, column\_statistic\_section):

for condition in column\_statistic\_section:

if condition == 'max':

self.CountMaxMin(df, column)

elif condition == 'maxLength':

self.CountLenOfRow(df, column)

elif condition == 'count\_unique\_val':

self.CountUnique(df, column)

elif condition == 'valuesCount':

self.CountQuantityofValues(df, column)

elif condition == 'nanCount':

self.CountNull(df, column)

# Метод подсчёта уникальных значений

def CountUnique(self, df, column):

ind = list(df[column].unique())

print(" Name column {}".format(column))

print(' Number of unique values: {}'.format(len(ind)))

# Метод подсчёт количества каждого значения в столбце

def CountQuantityofValues(self, df, column):

print(" Name column {}:".format(column))

print(" Quantity of {}".format(dict(Counter(df[column]).most\_common(100))))

# Метод подсчёта пустых строк

def CountNull(self, df, column):

x = df[column].isnull().sum()

print(" Name column {}".format(column))

print(" Quantity of null values: {}".format(x))

# Метод инциализирующий начало сбора статистики

def Statistics(self, df, config):

print(" Quantity of row in df: {}".format(len(df)))

for column in config["statistic"]:

self.DFStatistic(df, column, config["statistic"][column])

Модуль Run.py:

# -\*- coding: utf-8 -\*-

# Импорт библиотек

import json

import pandas as pd

# Импорт методов из других классов

from Validator import Validator

from Statistic import Statistic

# Метод инициализирующий запуск программы

def Run(configFile):

with open(configFile, encoding="utf-8") as json\_data:

config = json.loads(json\_data.read())

print("config have been read.")

print("data loading....")

df = pd.read\_csv(config["checkFile"], delimiter=config["delimiter"])

print("")

print("STATISTICS ------------------------------------------------------------------------------------")

stat = Statistic()

stat.Statistics(df, config)

print("")

print("COLUMNS VALIDATION ----------------------------------------------------------------------------")

validator = Validator(df, config)

validator.Validate()

print("")

print("end")

#if \_\_name\_\_ == '\_\_main\_\_':

# Run(input())

Модуль Validator.py:

# -\*- coding: utf-8 -\*-

# Импорт библиотек

import pandas as pd

from itertools import compress

from collections import Counter

class Validator:

ErrorsDF = None

Data = None

Config = None

# Инициация

def \_\_init\_\_(self, df, config):

self.Data = df

self.Config = config

self.ErrorsDF = df[0:0]

self.ErrorsDF['reason'] = []

# Метода распределения по подметодам валидации на наличие значений в друших справочниках

def Validate\_Is\_In(self, column, Is\_In):

self.Data = pd.read\_csv(self.Config["checkFile"], delimiter=';')

if Is\_In == 'Directory\_Catalog':

return self.Validate\_Catalog\_is\_In\_Directory\_Catalog(column)

elif Is\_In == 'Client\_Catalog':

return self.Validate\_CardID\_Is\_In\_ClientCatalog(column)

elif Is\_In == 'Shop\_catalog':

return self.Validate\_ShopID\_Is\_In\_ShopCatalog(column)

elif Is\_In == 'Item\_Catalog':

return self.Validate\_Article\_ID\_Is\_In\_Item\_Catalog(column)

return True, []

# Метод проверки на наличие UID магазина в справочнике магазинов

def Validate\_ShopID\_Is\_In\_ShopCatalog(self, column):

df\_shops = pd.read\_csv(self.Config["checkFile"], delimiter=';')

if self.Data['ShopID'] is df\_shops['ShopID']:

return True, []

else:

return False, []

# Метод проверки на наличие UID карты в справочнике клиентов

def Validate\_CardID\_Is\_In\_ClientCatalog(self, column):

df\_shops = pd.read\_csv(self.Config["checkFile"], delimiter=';')

if self.Data['DiscCardID'] is df\_shops['DiscCardID']:

return True, []

else:

return False, []

# Метод проверки на наличие UID артикула в справочнике товаров

def Validate\_Article\_ID\_Is\_In\_Item\_Catalog(self, column):

df\_shops = pd.read\_csv(self.Config["checkFile"], delimiter=';')

if self.Data['ICSKey'] is df\_shops['Article\_ID']:

return True, []

else:

return False, []

# Метода проверки на наличие Кода родительского раздела каталога в справочнике Каталогов

def Validate\_Catalog\_is\_In\_Directory\_Catalog(self, column):

df\_catalog = pd.read\_csv(self.Config["checkFile"], delimiter=';')

if self.Data['Catalog'] is df\_catalog['Code\_parent\_section\_catalog']:

return True, []

else:

return False, []

# Метод распределения по подметодам валидации на типы данных

def ValidateType(self, column, type, date\_mask):

if type == 'integer':

return self.ValidateTypeInt(column)

elif type == 'float':

return self.ValidateTypeFloat(column)

elif type == 'date':

return self.ValidateTypeDate(column, date\_mask)

return True, []

# Метод проверки на тип данных DateTime

def ValidateTypeDate(self, column, date\_mask):

self.Data[column] = pd.to\_datetime(self.Data[column], errors='coerce', format=date\_mask)

if self.Data[column].isnull().sum() > 0:

df\_with\_errors = self.Data[self.Data[column].isnull()]

return False, df\_with\_errors.index.tolist()

else:

return True, []

# Метод проверки на тип данных int

def ValidateTypeInt(self, column):

self.Data['col'] = pd.Series(self.Data[column] < 0)

res = list(self.Data['col'])

errors = list(compress(range(len(res)), res))

if len(errors) > 0:

return False, errors

else:

return True, []

# Метод проверки на тип данных float

def ValidateTypeFloat(self, column):

df = pd.DataFrame()

df['col'] = pd.to\_numeric(self.Data[column], errors='coerce')

df['numeric'] = df['col'].fillna(0)

df['float'] = df['numeric'].astype('float')

df['res'] = df['float'] == df['col']

res = list(df['res'])

errors = list(compress(range(len(res)), res))

errors\_list = list(set(list(range(len(res)))) - set(errors))

if len(errors\_list) > 0:

return False, errors\_list

else:

return True, []

# Метод проверки равенства фактической суммы и сумм полей оплаты

def ValidateAmount(self, column, right\_sum):

self.Data['amount'] = self.Data[column]

self.Data['cash'] = pd.to\_numeric(self.Data['SumPayCash'], errors='coerce').fillna(0)

self.Data['card'] = pd.to\_numeric(self.Data['SumPayCard'], errors='coerce').fillna(0)

self.Data['present\_card'] = pd.to\_numeric(self.Data['SumPayPresentCard'], errors='coerce').fillna(0)

self.Data['cupon'] = pd.to\_numeric(self.Data['SumPayCupon'], errors='coerce').fillna(0)

self.Data['bonnus'] = pd.to\_numeric(self.Data['BonusAmount'], errors='coerce').fillna(0)

self.Data['pay'] = self.Data['cash'] + self.Data['card'] + self.Data['present\_card'] + \

self.Data['cupon'] + self.Data['bonnus']

self.Data['result'] = self.Data[column] == self.Data['pay']

res = list(self.Data['result'])

errors = list(compress(range(len(res)), res))

errors\_list = list(set(list(range(len(res)))) - set(errors))

if len(errors\_list) > 0:

return False, errors\_list

else:

return True, []

def Validate\_Identity(self, column, identity):

if identity == 'Identity\_to\_Level':

return self.Validate\_Identity\_Catalogs\_to\_Level(column)

elif identity == 'Identity\_to\_Card':

return self.Validate\_Identity\_Card\_ID(column)

# Метод проверки на принадлежность карты к клиенту

def Validate\_Identity\_Card\_ID(self, column, column\_validate\_section):

cards = []

for row in self.Data[column]:

if row == '':

cards.append('F')

else:

cards.append('T')

self.Data['cards'] = cards

holders = []

for row in self.Data['DiscCardID']:

if row == '':

holders.append('F')

else:

holders.append('T')

self.Data['holders'] = holders

self.Data['result'] = self.Data['cards'] == self.Data['holders']

res = list(self.Data['result'])

errors = list(compress(range(len(res)), res))

errors\_list = list(set(list(range(len(res)))) - set(errors))

if len(errors\_list) > 0:

return False, errors\_list

else:

return True, []

# Метод проверки на уровень иерархии

def Validate\_Identity\_Catalogs\_to\_Level(self, column, column\_validate\_section):

code = []

for row in self.Data[column]:

if row == '':

code.append('empty')

else:

code.append('full')

self.Data['code'] = code

level = []

for row in self.Data['Level']:

if row == '' or row == 0:

level.append('empty')

else:

level.append('full')

self.Data['level'] = level

self.Data['result'] = self.Data['code'] == self.Data['level']

res = list(self.Data['result'])

errors = list(compress(range(len(res)), res))

errors\_list = list(set(list(range(len(res)))) - set(errors))

if len(errors\_list) > 0:

return False, errors\_list

else:

return True, []

# Метод проверки на преоблодание или равенство суммы (до скидки) над фактической суммой

def Validate\_Sum(self, column, column\_validate\_section):

self.Data['result'] = self.Data['Summa'] >= self.Data['Amount']

res = list(self.Data['result'])

errors = list(compress(range(len(res)), res))

errors\_list = list(set(list(range(len(res)))) - set(errors))

if len(errors\_list) > 0:

return False, errors\_list

else:

return True, []

# Метод проверки на длину строки

def ValidateLenOfCell(self, column, length):

arr = []

for row in self.Data[column]:

row = str(row)

if len(row) > length:

arr.append(self.Data.index[self.Data[column] == row].tolist())

if len(arr) > 0:

return False, arr

else:

return True

# Метод проверки на пустые значения

def ValidateNullCells(self, column, NotNull):

arr = []

self.Data = self.Data.fillna('')

for row in self.Data[column]:

if not row:

arr.append(self.Data.index[self.Data[column] == row].tolist())

if len(arr) > 0:

return False, arr[0]

else:

return True, []

def Validate\_Code\_Catalog(self, column, column\_validate\_section):

self.Data['result'] = self.Data[column] == self.Data['Code\_section\_catalog']

res = list(self.Data['result'])

errors\_list = list(compress(range(len(res)), res))

if len(errors\_list) > 0:

return False, errors\_list

else:

return True, []

# Метод проверки значений на уникальность

def ValidateUniqueValues(self, column, column\_validate\_section):

arr = set(self.Data[column])

counter = Counter(self.Data[column])

t = set([x for x in self.Data[column] if counter[x] > 1])

err = []

if len(arr) == len(self.Data[column]):

return True, []

else:

for item in t:

err.append(self.Data.index[self.Data[column] == item].tolist())

return False, err[0]

# Метод проверки значений на нахождение в допустимом диапазоне

def ValiadateAllowedValues(self, column, allowed\_values):

arr = []

a = list(set(self.Data[column].unique()) - set(allowed\_values))

for item in a:

arr.append(self.Data.index[self.Data[column] == item].tolist())

if len(arr) > 0:

return False, arr[0]

else:

return True, []

# Метод вывода отчёт в excel

def append\_df\_to\_excel(self, df, sheet\_name='Sheet1', startrow=None,

truncate\_sheet=False,

\*\*to\_excel\_kwargs):

from openpyxl import load\_workbook

if 'engine' in to\_excel\_kwargs:

to\_excel\_kwargs.pop('engine')

writer = pd.ExcelWriter('D:\\git\\validator\\reports\\Errors.xlsx', engine='openpyxl')

try:

writer.book = load\_workbook('D:\\git\\validator\\reports\\Errors.xlsx')

if startrow is None and sheet\_name in writer.book.sheetnames:

startrow = writer.book[sheet\_name].max\_row

if truncate\_sheet and sheet\_name in writer.book.sheetnames:

idx = writer.book.sheetnames.index(sheet\_name)

writer.book.remove(writer.book.worksheets[idx])

writer.book.create\_sheet(sheet\_name, idx)

writer.sheets = {ws.title: ws for ws in writer.book.worksheets}

except FileNotFoundError:

pass

if startrow is None:

startrow = 0

df.to\_excel(writer, sheet\_name, startrow=startrow, \*\*to\_excel\_kwargs)

writer.save()

# Основной метод валидации, инициирующий начало остальных

def Validate(self):

checked = None

resColumns = ['column', 'condition', 'info', 'passed', 'errors']

resDF = pd.DataFrame([], columns=resColumns)

for column in self.Config["columns\_for\_validation"]:

try:

colSection = self.Config["columns\_for\_validation"][column]

for key in colSection:

sec = colSection[key]

errors = []

if key == 'type':

checked = True

if 'date\_mask' in colSection:

date\_mask = colSection['date\_mask']

else:

date\_mask = '%yyyy-%MM-%dd'

res, errors = self.ValidateType(column, sec, date\_mask)

elif key == 'allowed\_values':

checked = True

res, errors = self.ValiadateAllowedValues(column, sec)

elif key == 'unique':

checked = True

res, errors = self.ValidateUniqueValues(column, sec)

elif key == 'length':

checked = True

res, errors = self.ValidateLenOfCell(column, sec)

elif key == 'NotNull':

checked = True

res, errors = self.ValidateNullCells(column, sec)

elif key == 'right\_sum':

checked = True

res, errors = self.ValidateAmount(column, sec)

elif key == 'Bigger\_Than\_Amount':

checked = True

res, errors = self.Validate\_Sum(column, sec)

elif key == 'equality':

checked = True

res, errors = self.Validate\_Code\_Catalog(column, sec)

elif key == 'Is\_In':

checked = True

res, errors = self.Validate\_Is\_In(column, sec)

elif key == 'Identity':

checked = True

res, errors = self.Validate\_Identity(column, sec)

#self.ErrorsDF.append(errors)

if checked:

resDF.loc[len(resDF)] = [column, key, sec, res, errors]

except Exception as ex:

print('Error while processing {} {}'.format(column, colSection))

print(ex)

print(resDF.head(100))

if self.Config["checkFile"] == 'D:\\git\\validator\\testData\\csv\\catalog\_test.csv':

self.append\_df\_to\_excel(resDF, sheet\_name="Catalog")

elif self.Config["checkFile"] == 'D:\\git\\validator\\testData\\csv\\clientfull\_test.CSV':

self.append\_df\_to\_excel(resDF, sheet\_name="Clients")

elif self.Config["checkFile"] == 'D:\\git\\validator\\testData\\csv\\items\_test.csv':

self.append\_df\_to\_excel(resDF, sheet\_name="Items")

elif self.Config["checkFile"] == 'D:\\git\\validator\\testData\\csv\\sales\_test.csv':

self.append\_df\_to\_excel(resDF, sheet\_name="Sales")

elif self.Config["checkFile"] == 'D:\\git\\validator\\testData\\csv\\shop\_test.csv':

self.append\_df\_to\_excel(resDF, sheet\_name="Shops")

elif self.Config["checkFile"] == 'D:\\git\\validator\\testData\\csv\\delivery\_test.csv':

self.append\_df\_to\_excel(resDF, sheet\_name="Status\_Delivery")

CatalogConfig.json:

{

"delimiter": ";",

"most\_common": 1,

"checkFile": "D:\\git\\validator\\testData\\csv\\catalog\_test.csv",

"statistic":

{

},

"columns\_for\_validation":

{

"Code\_parent\_section\_catalog":

{

"length": 50,

"Identity\_to\_Level": "true",

"equality": "false"

},

"Code\_section\_catalog":

{

"length": 50,

"unique": "true"

},

"Name\_section\_catalog":

{

"length": 255

},

"Level":

{

"type": "integer"

},

"Delete\_point":

{

"allowed\_values": [0, 1]

}

}

}

ClientsConfig.json:

{

"delimiter": ";",

"most\_common": 1,

"checkFile": "D:\\git\\validator\\testData\\csv\\clientfull\_test.CSV",

"statistic":

{

"DiscCardID":["max", "min"],

"SEX":["maxLength", "valuesCount"],

"followInfoSms":["valuesCount"],

"followAdvertising":["valuesCount"],

"StatusCard":["valuesCount"],

"followInfoEmail":["valuesCount"],

"DiscCardHolderID":["count\_unique\_val"]

},

"columns\_for\_validation":

{

"DiscCardHolderID": {

"length": 30,

"unique": "true",

"NotNull": "true"

},

"DiscCardID":

{

"length": 15,

"type" : "integer",

"NotNull": "true"

},

"DateBegin":

{

"type" : "date",

"date\_mask" : "%Y-%m-%d",

"NotNull": "true"

},

"BirthDate":

{

"type" : "date",

"date\_mask" : "%Y-%m-%d"

},

"SEX":

{

"allowed\_values":["муж", "жен"]

},

"followAdvertising":

{

"allowed\_values":[1,0]

},

"StatusCard":

{

"allowed\_values":[1,2]

},

"followInfoSms":

{

"allowed\_values":[1,0]

},

"followInfoEmail":

{

"allowed\_values":[1,0]

},

"ShopID" :

{

"NotNull": "true",

"length": 30,

"Is\_In": "Shop\_Catalog"

}

},

"max\_errors": 100

}

ItemsConfig.json:

{

"delimiter": ";",

"most\_common": 1,

"checkFile": "D:\\git\\validator\\testData\\csv\\items\_test.csv",

"statistic":

{

},

"columns\_for\_validation":

{

"Catalog":

{

"NotNull": "true",

"length": 50,

"Is\_In": "Directory\_Catalog"

},

"Article\_ID":

{

"NotNull": "true",

"unique": "true",

"length": 50

},

"Item\_code":

{

"length": 50

},

"Item\_name":

{

"length": 100

},

"Item\_full\_name":

{

"length": 255

},

"Brand":

{

"length": 100

},

"DT":

{

"type": "date",

"date\_mask": "%Y-%m-%d"

},

"Delete\_point":

{

"allowed\_values": [1, 0]

}

}

}

saleConfig.json:

{

"delimiter": ";",

"most\_common": 1,

"checkFile": "D:\\git\\validator\\testData\\csv\\sales\_test.csv",

"statistic":

{

"DiscCardHolderID": ["count\_unique\_val"],

"Summa": ["max", "min"],

"Amount": ["max", "min"],

"AccruedBonus": ["max", "min"],

"BonusAmount": ["max", "min"],

"SumPayCash": ["max", "min"],

"SumPayCard": ["max", "min"],

"SumPayPresentCard": ["max", "min"],

"SumPayCupon": ["max", "min"],

"SellerID": ["nanCount"],

"TypeDelivery": ["valuesCount"],

"SaleChannel": ["valuesCount"]

},

"columns\_for\_validation": {

"DT":

{

"type": "date",

"date\_mask": "%Y-%m-%d"

},

"Qty": {

"type": "float",

"NotNull": "true"

},

"PublicityID": {

"type": "string",

"length": 25

},

"DiscCardHolderID": {

"type": "string",

"length": 30,

"NotNull": "true",

"Identity\_to\_Card": "true",

"Same\_Cheque\_ID": "false"

},

"DiscCardID":

{

"type": "string",

"length": 30,

"NotNull": "true",

"Is\_In": "Client\_Catalog"

},

"ShopID":

{

"type": "string",

"length": 30,

"NotNull": "true",

"Is\_In": "Shop\_catalog"

},

"ICSKey":

{

"type": "string",

"length": 30,

"NotNull": "true",

"Is\_In": "Item\_Catalog"

},

"Summa":

{

"type": "float",

"Bigger\_Than\_Amount": "true"

},

"Amount":

{

"type": "float",

"right\_sum": "true"

},

"AccruedBonus":

{

"type": "float",

"Client\_ID\_Is\_Noy\_Null": "true",

"Card\_ID\_Is\_Noy\_Null": "true"

},

"BonusAmount":

{

"type": "float"

},

"SumPayCash":

{

"type": "float"

},

"SumPayCard":

{

"type": "float"

},

"SumPayPresentCard":

{

"type": "float"

},

"SumPayCupon":

{

"type": "float"

},

"SaleChannel":

{

"allowed\_values": ["",1, 2, 3]

},

"TypeDelivery":

{

"allowed\_values": ["",1, 2, 3]

},

"SellerID": {

"type": "string",

"length": 50

},

"ChequeID": {

"unique": "true",

"type": "string",

"length": 50,

"NotNull": "true"

}

}

}

shopsConfig.json:

{

"delimiter": ";",

"most\_common": 1,

"checkFile": "D:\\git\\validator\\testData\\csv\\shop\_test.csv",

"statistic":

{

},

"columns\_for\_validation":

{

"ShopID":

{

"length": 30,

"unique": "true",

"NotNull": "true"

},

"Shop\_Name":

{

"length": 255

},

"Address":

{

"length": 255

},

"City":

{

"NotNull": "true",

"length": 255

},

"Region":

{

"length": 255

},

"Status":

{

"NotNull": "true",

"allowed\_values": [1, 0]

},

"Type":

{

"NotNull": "true",

"length": 50

}

}

}

statusConfig.json:

{

"delimiter": ";",

"most\_common": 1,

"checkFile": "D:\\git\\validator\\testData\\csv\\delivery\_test.csv",

"statistic":

{

},

"columns\_for\_validation":

{

"Client\_ID":

{

"length": 30,

"NotNull": "true"

},

"Card\_ID":

{

"length": 15,

"NotNull": "true"

},

"Recommend":

{

"length": 255

},

"Status\_Delivery":

{

"length": 30

},

"DT":

{

"type": "date",

"date\_mask": "%Y-%m-%d"

},

"Type":

{

"length": 30

}

}

}