## TestTableParser

Класс функций для обработки файла

* IsVerticalOrientation – Вертикально или горизонтально ориентирована таблица
  + Boolean: True/False
* index\_rows – список индексов строк
  + List []
* columns\_types – список типов колонок, словарь, ключ – номер, значение – тип
  + KeyValuePair {}
* mistake\_fix – список словарей, каждый словарь – 7 пар ключ-значение с информацией об положении ошибки, ее характере, и вероятности
  + [{…….}]
* start\_point – индекс строки начиная с которой в таблице идут данные
  + int, номер. По дефолту 0

### Определение ориентации таблицы (гризонтальная/вертикальная)

По умолчанию считаем все таблицы вертикальными. Функция выполняет проверку входной таблицы на вертикальность, возвращает соответственно, True или False

* name
  + orientation\_determine
* input
  + DataFrame, исходная таблица
* Output
  + True/False
* Realization
  + Если количество столбцов, содержащих числовые значения, без пропусков, больше, чем число таких же строк, то таблицу считаем горизонтальной, и в дальнейшем транспонируем. Соответственно, возвращаем False, иначе True
* Example
  + Any
* Implementation

def orientation\_determine(self, df):

        list\_columns = []

        for j in range(df.shape[1]):

            list\_columns.append(pd.to\_numeric(df.iloc[:, j], errors='coerce').dropna())

        clear\_df = pd.concat(list\_columns, axis=1)

        if clear\_df.shape[0] > clear\_df.shape[1]:

            self.IsVerticalOrientation = True

            return self.IsVerticalOrientation

        else:

            self.IsVerticalOrientation = False

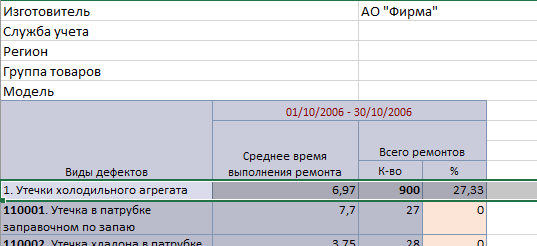
            return self.IsVerticalOrientation

* Test-case
  + ‘Table3.xlsx’

### Определение поля данных

В таблице присутствуют заголовки, имена столбцов, строк. Функция определяет строку, начиная с которой в таблице, если она вертикальная, идут данные в столбцах.

* Name
  + Find\_start\_point
* Input
  + DataFrame, обработанная исходная таблица. Получаем ее из исходной с помощью to\_numeric\_func
* Output
  + int, номер. По дефолту 0
* Realization
  + Возвращаем индекс первой строки, количество непустых ячеек численного формата в которой больше двух.
* Example



* Implementation

def find\_start\_point(self, clear\_df):

        for i in range(clear\_df.shape[0]):

            if len(clear\_df.iloc[i, :].dropna()) > 2:

                self.start\_point = i

                break

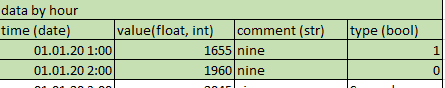
        return self.start\_point

* Test-case
  + “Table1”, Any

### Определение типов столбцов

Функция определяет тип данных в столбцах таблицы

* Name
  + define\_type\_of\_cell
* input
  + DataFrame – исходная таблица
  + DataFrame – обработанная исходная таблица
  + Int - индекс строки начиная с которой в таблице идут данные
* Output
  + KeyValuePair {} - список типов колонок, словарь, ключ – номер, значение – тип
* Realization
  + Проверяем данные по столбцам на пять заранее заданных типов данных: целый, строковый, с плавающей запятой, дата (время), логический. Сперва проверяем, является ли ячейка числом, и если да, то является ли целым. Если числом не является, то проверяем, является ли датой (временем), и если нет, то проверяем на логический. Все остальное считаем строкой
* Example



* Implementation

def define\_type\_of\_cell(self, clear\_df, df\_data, start\_point):

        for j in range(clear\_df.shape[1]):

            baff = clear\_df.iloc[:, j].dropna()

            if baff.sum() != 0:

                if baff.dtype == pd.Float64Dtype():

                    self.columns\_types[j] = 'float'

                else:

                    self.columns\_types[j] = 'int'

            else:

                if type(self.parse(df\_data.iloc[:, j].astype(str))[0][start\_point]) == type(pd.to\_datetime('31.01.2002')):

                    self.columns\_types[j] = 'date'

                elif baff.dtype == pd.BooleanDtype():

                    self.columns\_types[j] = 'bool'

                else:

                    self.columns\_types[j] = 'str'

        return self.columns\_types

### define\_breakline

* Input
  + DataFrame– обработанная исходная таблица
  + KeyValuePair List {} – типы столбцов таблицы
* Output
  + List – список индексов пропущенных строк (разрыв таблицы)

### create\_list\_of\_table

* Input
  + DataFrame – исходная таблица
  + List – список breakline-ов
  + Int – start point, индекс первой строки данных
* Output
  + List – список индексов строк “рабочей” таблицы (без breakline-ов)

### Определение заголовка таблицы

* Name
  + Create\_header
* Input
  + DataFrame
  + Int
  + DataFrame (numeric)
* Output
  + DataFrame
* Realization
  + Возвращает список (под)заголовков, взятых из таблицы до стартовой точки
* Example
* Implementation

def create\_header(self, df, start\_point, df\_data):

        cols = []

        count = 1

        topic = df.iloc[:start\_point, :].fillna(method='ffill', axis=1)

        list\_topic = []

        for i in range(topic.shape[1]):

            list\_topic.append("\_\_".join(list(topic.iloc[:, i].astype(str))).replace("\_\_nan", "").replace("nan\_\_", ""))

        for column in list\_topic:

            if column in cols:

                cols.append("{0}\_{1}".format(str(column), str(count)))

                count += 1

                continue

            cols.append(column)

        df\_data.columns = cols

        return df\_data

* Test case

Возвращает список (под)заголовков, взятых из таблицы до стартовой точки

### to\_numeric\_func

* Input DataFrame исходный
* Output DataFrame обрабтанный

Возвращает отредактированную исходную таблицу. Все значения переведены в числовые

### json\_creator

создает файл с метаданными по таблице

### Определение ошибок ввода

* Name
  + Find\_errors
* Input
  + DataFrame – срез столбца исходной таблицы
  + Int – индекс (строки)
* Output
  + 2D List [индекс сроки, {список KeyValuePair сведений об ошибках в строке}]
* Realization
  + Проверка строк на предмет путаницы в схожих символах (хардкод)
* Example



* Implementation

def find\_errorss(self, s, j):

        mistake = {'o': '0', 'l': '1', 'b': '6', 'g': '9', 'q': '9', 't': '7', 'v': '5', 'f': '7', 'z': '2', 'e': '3',

                   's': '5'}

        for key, val in mistake.items():

            if len(s[s.astype(str).str.contains(key).dropna()].index.to\_list()) != 0:

                for i in range(len(s[s.astype(str).str.contains(key)].index.to\_list())):

                    self.mistake\_fix.append({'row': s[s.astype(str).str.contains(key)].index.to\_list()[i],

                                         'column': j,

                                         'new value': val,

                                         'old value': key,

                                         'probability': 99.9,

                                         'user approve': False,

                                         'comment': False

                                         })

                s = s.astype(str).str.strip().str.replace(key, val, regex=True)

            else: continue

        return [s, self.mistake\_fix]

Проверка строк на предмет путаницы в схожих символах (хардкод)

### change\_valuee

* Input
  + DataFrame
  + List список типов столбцов
* Output
  + 2D list

Возвращается двумерный список, где каждой ячейке вх. данных ставится в соответствие информация об ошибке/её отсутствии

### all\_action

* Input
  + DataFrame
* Output
  + DataFrame
  + Json

Функция, выполняющая все преобразования. На выходе – преобразованная таблицы и файл метаданных