

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Новосибирский национальный исследовательский государственный университет»
(Новосибирский государственный университет, НГУ)
Структурное подразделение Новосибирского государственного университета –
Высший колледж информатики Университета (ВКИ НГУ)
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Направление подготовки: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Образовательная программа: 09.03.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

**СОЗДАНИЕ БИБЛИОТЕКИ АЛГОРИТМОВ ДЛЯ
СТАТИСТИЧЕСКОГО АНАЛИЗА ДАННЫХ КЛИНИЧЕСКИХ
ИССЛЕДОВАНИЙ В ПАРАЛЛЕЛЬНЫХ ГРУППАХ**

утверждена приказом по ВКИ НГУ № 02/3-204 от «27» апреля 2017г.

Кошкаревой Софьи Владимировны, группа 14214
(фамилия, имя, отчество студента)

_____ (подпись студента)

«К защите допущена»

Заведующий кафедрой, к.ф.-м.н
(ученая степень, звание)

Попов Л.К., / _____
(ФИО) / (подпись)

«___» _____ 20__ г.

Руководитель ВКР

к.ф.-м.н,
(ученая степень, звание)

Лукинов В.Л., / _____
(ФИО) / (подпись)

«___» _____ 20__ г.

Дата защиты: «___» _____ 20__ г.

Новосибирск
2017

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	4
ВВЕДЕНИЕ	5
1 ПОСТАНОВКА ЗАДАЧИ	6
1.1 Описание предметной области	6
1.1 Формулировка задачи	8
1.2 Функциональные требования	8
1.3 Нефункциональные требования	9
1.4 Характеристики выбранных программных средств	9
2 РЕАЛИЗАЦИЯ	11
2.1 Алгоритмы решения задач	11
2.1.1 Опечатки	11
2.1.2 Выбросы	12
2.1.3 Генерация отчетов	13
2.2 Объектно-ориентированная модель	14
2.2.1 Файлы	14
2.2.2 Типы значений столбцов таблицы	16
2.2.3 Типы ошибок	18
2.2.4 Описание реализации поиска опечаток	20
2.2.4.1 Метод поиска опечаток для дискретных значений	20
2.2.4.2 Метод поиска опечаток для непрерывных значений	22
2.2.4.3 Метод поиска опечаток для дат	24
2.2.5 Реализация поиска выбросов	26
2.2.6 Реализация поиска неупорядоченных дат	29
2.2.7 Создание сводной таблицы значений	30
3 РАЗРАБОТКА БИБЛИОТЕКИ	33
4 ОТЛАДКА И ТЕСТИРОВАНИЕ	35
5 РЕЗУЛЬТАТЫ ОБРАБОТКИ	36
ЗАКЛЮЧЕНИЕ	42
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	43

ПРИЛОЖЕНИЕ А	45
--------------------	----

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Выброс	Значение, которое слишком велико или слишком мало по сравнению с большинством других имеющихся значений.
Параллельные группы	Это две или более группы испытуемых, одна или более из которых получают исследуемый препарат, а другая группа является контрольной.
Медиана	Такое число, что вероятность получить значение случайной величины справа от него равна вероятности получить значение слева от него.
Математическое ожидание	Это среднее значение случайной величины.
Дисперсия	Среднее значение квадрата отклонения случайной величины от её математического ожидания.
Среднеквадратическое отклонение	Значение, равное квадратному корню из дисперсии случайной величины.
Нормальное распределение	Это распределение вероятностей, совпадающее с функцией Гаусса.
Квантиль	Это значение, которое заданная случайная величина не превышает с фиксированной вероятностью.
ИКР	Интерквартильный размах.
Интерквартильный размах	Разность между третьим и первым квантилями.

ВВЕДЕНИЕ

Целью данной работы является разработка программного обеспечения, предназначенного для проведения статистического анализа данных, полученных в результате клинических исследований.

На протяжении своего развития медицинское сообщество всегда старалось найти более эффективные способы лечения и диагностики болезней. Первоначальные методы были неэффективны из-за применения метода проб и ошибок и интуитивных обобщений. Для решения этой проблемы в медицине сформировалась новая область – доказательная медицина [1].

Доказательная медицина подразумевает такой подход к медицинской практике, при котором каждое решение, относящееся к выбору метода лечения, обязано иметь научное обоснование. Оно основывается на данных, полученных в ходе четко спланированного и документированного исследования, использующего методы статистического анализа.

Опечатки и пропуски отдельных значений в данных исследований являются постоянной проблемой [2]. Поэтому, прежде чем начать применять статистические методы, обрабатываемые данные следует привести к приемлемому для обработки виду. Для этого необходимо идентифицировать возможные проблемы, а в дальнейшем, либо удалить некорректные данные, либо заменить имеющиеся пропуски и опечатки разумными значениями, что и было реализовано в данной работе. Еще одной важной проблемой является наличие выбросов. Под «выбросом» понимается значение, которое слишком велико или слишком мало по сравнению с большинством других имеющихся значений.

Разработанная библиотека решает проблему подготовки входных данных для статистического анализа в автоматическом режиме. Библиотека сводит к минимуму затраченное на это время, а также позволяет биостатистикам анализировать данные исследования в удобной форме.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Описание предметной области

В большинстве случаев клинические исследования проводятся в параллельных группах. Параллельные группы – это две или более группы испытуемых, одна или более из которых получают исследуемый препарат, а другая группа является контрольной. Для достижения статистической достоверности испытуемые распределяются по группам методом случайной выборки [3].

При подготовке данных к статистическому анализу клинических исследований необходимо выполнять следующие рутинные процедуры:

- Поиск пропущенных значений (незаполненных полей).
- Исправление опечаток.
- Обнаружение выбросов.
- Проверка данных на условие нормальности распределения.
- Поиск неупорядоченных дат.

На данный момент выполнение этих процедур происходит в ручном режиме. Такой подход является неэффективным из-за человеческой невнимательности. При таком подходе невозможно обработать большой объем данных за разумное время.

Поэтому программная проверка существенно ускорит дальнейший анализ, а так же окажется более эффективной по сравнению с проверкой, выполняемой в ручном режиме.

Методы статистического анализа делятся на два вида:

- Параметрические.
- Непараметрические.

Параметрические методы имеют более высокую точность и эффективность по сравнению с непараметрическими, но имеют ограничения на входные данные – данные должны быть нормально распределены.

Нормальное распределение (или распределение Гаусса) – это распределение вероятностей, совпадающее с функцией Гаусса, вычисляемой по формуле, представленной ниже [4].

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

где

μ – математическое ожидание (среднее значение),

σ – среднеквадратическое отклонение.

Математическое ожидание – это среднее значение случайной величины [5].

Дисперсией случайной величины называют математическое ожидание квадрата отклонения случайной величины от ее математического ожидания [6].

Среднеквадратическим отклонением случайной величины называется квадратный корень из дисперсии этой величины [7].

Стандартным нормальным распределением называется нормальное распределение с математическим ожиданием $\mu = 0$ и стандартным отклонением $\sigma = 1$. Стандартное нормальное распределение отображено на рисунке 1 зеленой линией.

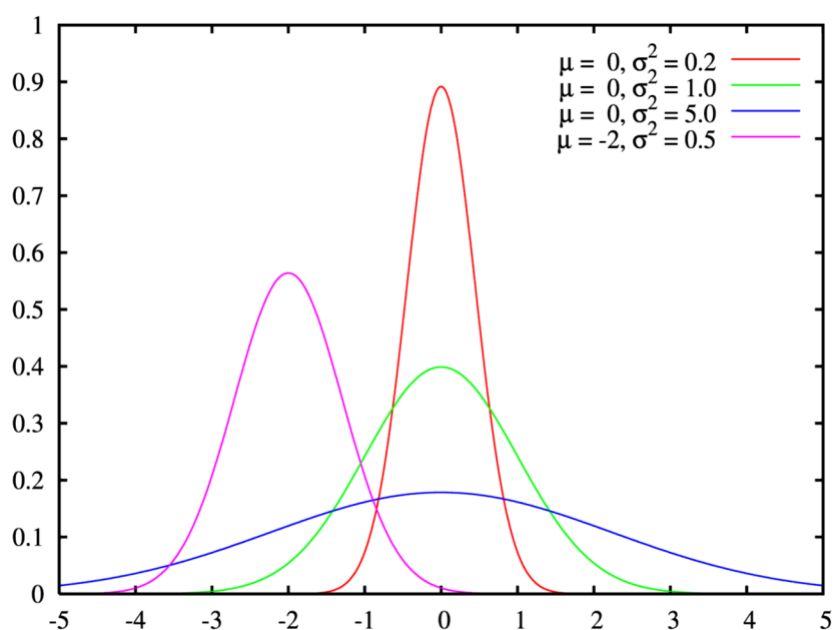


Рисунок 1 – Нормальное распределение

Перед применением статистических моделей нужно удостовериться, что данные имеют нормальное распределение.

Создание библиотеки алгоритмов статистического анализа данных для клинических исследований в параллельных группах, позволит существенно сократить время проведения статистического анализа и поможет проводить более качественные исследования в короткие сроки.

1.1 Формулировка задачи

Целью работы являлась разработка библиотеки языка R с инструкцией по эксплуатации, предназначенной для проведения статистического анализа данных клинических исследований в параллельных группах.

Программирование осуществлялось на языке R, с использованием следующих библиотек:

- base.
- methods.
- utils.
- grDevices.
- plyr.
- xlsx.
- sm.
- nortest.
- devtools.
- roxygen2.

Для обеспечения возможности возврата к определённым старым версиям разработки, использовалась система контроля версий git и репозиторий на сервере GitHub.

1.2 Функциональные требования

В рамках дипломной работы были поставлены следующие задачи:

- Поиск пропущенных значений (незаполненных полей).
- Поиск опечаток.
- Поиск выбросов.
- Проверка на упорядочение дат.
- Исследование нормальности распределения данных различными статистическими критериями.

1.3 Нефункциональные требования

В рамках дипломной работы были определены следующие нефункциональные требования:

- Ввод данных должен осуществляться в виде Excel-файла или внутренней структуры языка R–таблиц данных (data.frame).
- Результаты работы должны записываться в Excel-файл.
- Реализация методами ООП, используя объектную модель S4 в языке R.
- Использование системы контроля версий Git в связке с сервером GitHub.
- Тестирование созданной библиотеки на реальных данных.

1.4 Характеристики выбранных программных средств

Для решения задачи был выбран язык R и среда разработки RStudio.

R – язык программирования высокого уровня, предназначенный для статистической обработки данных с упором на визуализацию и воспроизводимость. Он так же является свободной кроссплатформенной программной средой вычислений с открытым исходным кодом в рамках проекта GNU. R – интерпретируемый язык с интерфейсом командной строки. Он сочетает в себе процедурное, функциональное и объектно-ориентированное программирование [8].

На сегодняшний день R является широко известным среди свободно распространяемых систем статистического анализа. Он обладает хорошей расширяемостью с помощью библиотек, содержащих набор специфических функций. В среде R реализованы многие статистические методы: линейные и нелинейные модели, проверка статистических гипотез, анализ временных рядов, классификация, кластеризация, графическая визуализация. Еще одной из особенностей языка является поддержка графических возможностей, позволяющая визуализировать данные в виде различных графиков и диаграмм.

2 РЕАЛИЗАЦИЯ

2.1 Алгоритмы решения задач

Различные статистические методы по-разному относятся к наличию выбросов во входных данных. Наличие выбросов может сделать использование определенных статистических моделей *невозможным*, и в то же время, никак не сказаться на результатах других [9].

Именно поэтому перед проведением статистического анализа необходимо выполнять проверку начальных данных на валидность.

Валидизация данных – это процесс обнаружения и исправления ошибок.

Для каждого рода ошибок был разработан свой алгоритм обнаружения, о которых будет рассказано ниже.

2.1.1 Опечатки

Опечатки – это некорректно введенные пользователем данные. Примером опечаток может служить:

- Данные, не входящие в набор определенных допустимых значений.
- Наличие буквенных символов в числовых данных.
- Неправильные разделители между числами в записи дат.
- Лишние пробелы в записи десятичных дробей.

На рисунке 2 изображена схема алгоритма поиска опечаток для данных, имеющих набор определенных допустимых значений. Например, записи о тяжести состояния пациента (только следующие значения: «удовлетворительное», «средней тяжести», «тяжёлое», «крайне тяжёлое»).

Остальные виды опечаток определяются с помощью шаблонов, заданных регулярными выражениями.

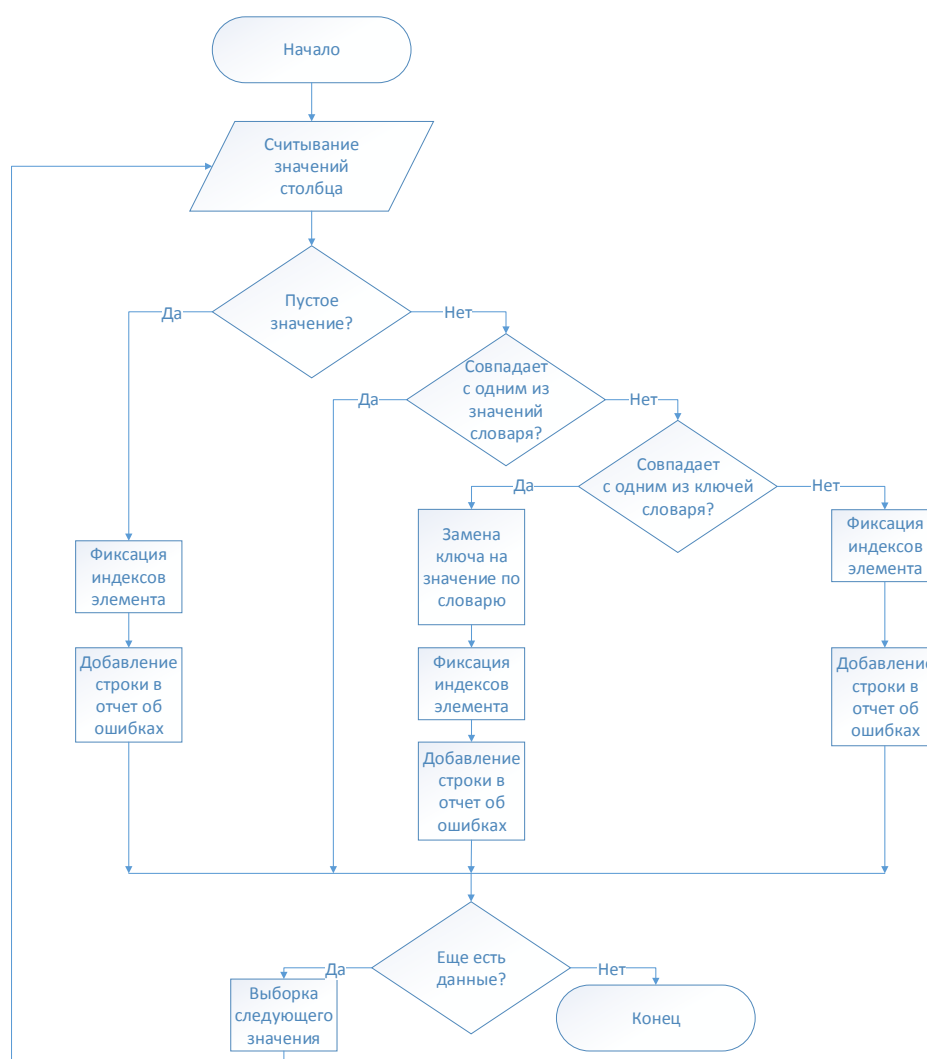


Рисунок 2 – Схема работы алгоритма поиска опечаток для данных с набором допустимых значений

2.1.2 Выбросы

Под «выбросом» понимается значение, которое слишком велико или слишком мало по сравнению с большинством других имеющихся значений. К примеру, средний вес большинства участников исследования – 75 кг. Но в исследовании также приняли участие два человека весом 55 кг и 110 кг, именно эти значения будут выбросами.

Причины появления выбросов могут быть различными:

- Из-за ошибки измерения.
- Из-за необычной природы входных данных. Например, если наугад измерять температуру предметов в комнате, то большинство

полученных значений будет лежать в диапазоне от 18 до 22 °С, но радиатор отопления будет иметь температуру в 70°.

- Выбросы могут быть и частью распределения.

Для обнаружения выбросов была использована функция `boxplot.stats()`, производная графической функции высокого уровня `boxplot()`, которая служит для построения диаграмм размахов. А `boxplot.stats()`, в свою очередь, используется для сбора статистики, необходимой для создания диаграмм размахов. Подробный механизм реализации описан в пункте 2.2.5.

К выбросам всегда следует относиться внимательно. Они вполне могут оказаться «нормальными» для исследуемой совокупности, и поэтому требуют дополнительного изучения причин их появления. Поэтому производится только идентификация таких значений, а не поспешное удаление и исключение их из статистического анализа.

2.1.3 Генерация отчетов

Библиотека позволяет создавать пользовательские отчеты об ошибках нескольких типов:

- Текстовый файл, в который построчно записываются сообщения о найденных некорректных данных. Текст сообщения содержит тип ошибки, позицию некорректного значения в таблице данных, само значение.
- Excel-файл с несколькими стилями, которые применяются для наглядного отображения найденных ошибок.
- Сводная таблица данных в виде Excel-файла, которая содержит различные значения каждого столбца исходных данных и частоту встречаемости этих значений. Она дает верное представление о содержимом входных данных перед началом их обработки, а также помогает в определении допустимого набора значений для исправления опечаток.

2.2 Объектно-ориентированная модель

Поставленные задачи были реализованы с использованием объектно-ориентированной (ООП) модели языка R. Для решения поставленных задач были разработаны три независимых структуры классов:

- Файлы.
- Столбцы таблицы.
- Типы ошибок.

Подробнее каждая структура описана в следующих пунктах.

2.2.1 Файлы

В процессе выявления различных ошибок требуется обеспечить ввод и вывод различных данных. Для этого была разработана структура классов «Файлы», которая представлена на рисунке 3.

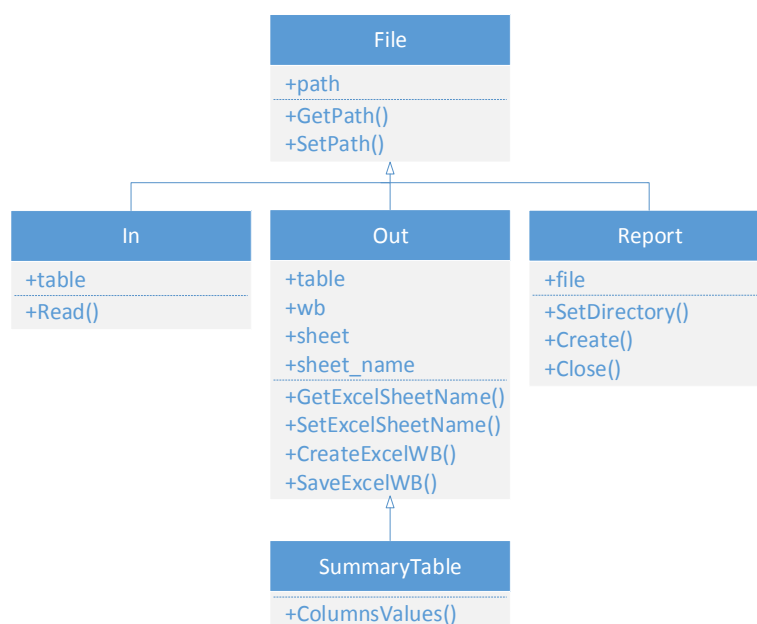


Рисунок 3 – Структура классов «Файлы»

Класс-родитель *File* имеет одно поле *path* и два метода для взаимодействия с ним. Метод *SetPath()* устанавливает полный путь к файлу, а метод *GetPath()* служит для получения текущего. У класса *File* существует три

дочерних класса: *In*, *Out* и *Report*. Каждый из них наследует родительские методы и поле *path*, значение которого для каждого класса будет разным.

Класс *In* имеет метод *Read()*, реализующий считывание начальных данных из Excel-файла, и поле *table*, куда помещается результат считывания в виде внутренней структуры языка R, таблицы данных (*data.frame*).

Результат исправления опечаток в исходных данных присваивается полю *table* класса *Out*.

Также, в классе *Out* реализованы методы для записи результатов в Excel и созданы необходимые для этих методов поля. Поля *wb* и *sheet* являются объектами класса *jobjRef* из библиотеки *rJava*, которая, в свою очередь, используется библиотекой *xlsx* для связи Java и R. Поле *sheet_name* содержит название листа для новой рабочей книги Excel. Назначить новое название можно используя *SetExcelSheetName()*, а чтобы узнать текущее – метод *GetExcelSheetName()*.

Метод *CreateExcelWB()* нужен для создания новой рабочей книги с именованным листом, и добавления на него итоговой таблицы данных, к которой будут применены стили. Также в этом методе создается новая пустая строка в шапке таблицы для того, чтобы разместить там обозначения различных типов ошибок.

Метод *SaveExcelWB()* сохраняет рабочую книгу, используя в качестве полного пути содержимое поля *path* объекта *Out*. Перед сохранением файла устанавливается автоподбор ширины столбцов и закрепляется первая строка таблицы (легенда). Методы *CreateExcelWB()* и *SaveExcelWB()* являются «оберткой» для работы с библиотекой *xlsx*.

Для работы с текстовым файлом и записью в него сообщений для пользователя был создан класс *Report*. У него есть поле *file*, которое используется для создания файла-отчета. Полный путь к файлу генерируется методом *setDirectory()*, который добавляет к указанной при вызове метода директории строку «Report_» и текущую дату и время. Метод *Create()* создает

по указанному полному пути файл и открывает соединение для записи в текстовом режиме, а метод *Close()* закрывает это соединение.

Сводная таблица должна содержать названия столбцов исходной таблицы, их значения и частоту встречаемости каждого из значений. Для создания сводной таблицы был создан класс *SummaryTable*, являющийся потомком класса *Out*, и метод *ColumnsValues()*, механизм работы которого описан в пункте 2.2.7.

2.2.2 Типы значений столбцов таблицы

Структура данных таблицы, с которой работает библиотека, может состоять из четырех типов переменных:

- Дата.
- Непрерывные переменные.
- Дискретные переменные.
- Категориальные переменные.

В виде даты в результатах исследования может указываться время измерения различных показателей пациента, время его поступления в клинику и время выписки. Важно следить за тем, чтобы даты повторных измерений сохраняли упорядоченность, т.е. чтобы дата повторного измерения была больше даты первичного.

Непрерывные переменные могут принимать любые численные значения, которые естественным образом упорядочены на числовой оси (например, рост, вес).

Дискретные переменные могут принимать счётное множество упорядоченных значений, которые могут просто обозначать целочисленные данные или ранжировать данные по степени проявления на упорядоченной ранговой шкале (клиническая стадия опухоли, тяжесть состояния пациента).

Категориальные переменные являются неупорядоченными и используются для качественной классификации (пол, цвет глаз, место жительства); в частности, они могут быть бинарными (дихотомическими) и

иметь категорические значения: 1 / 0, да / нет, имеется / отсутствует. Поэтому каждый из четырех типов значений описывает свой класс, и для каждого из них реализован свой метод поиска ошибок. Структура классов «Столбцы таблицы» представлена на рисунке 4.

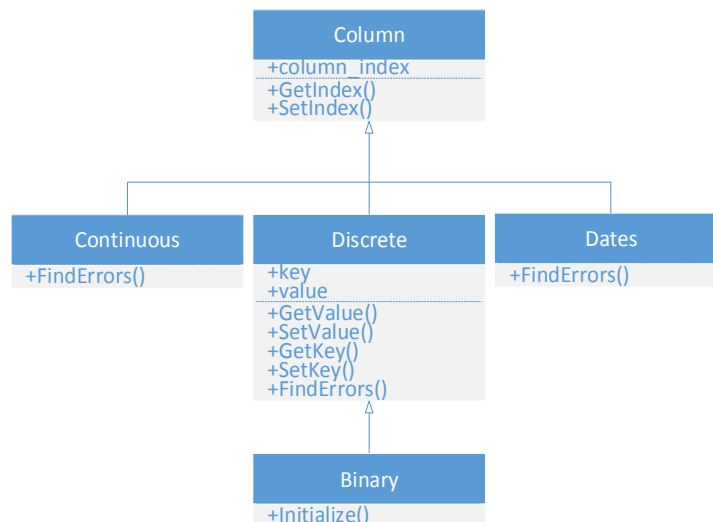


Рисунок 4 – Структура классов «Столбцы таблицы»

Родительский класс *Column* имеет поле `column_index`, в котором хранится номер столбца в исследуемой таблице, и два метода: `GetColumnIndex()`, `SetColumnIndex()` для задания нового значения этого поля и получения текущего. Данные методы доступны во всех четырех дочерних классах.

Объекты класса *Continuous* описывают столбцы таблицы, содержащие непрерывные переменные. Реализация метода `FindErrors()` для объектов класса *Continuous* состоит из поиска опечаток и поиска выбросов.

Объекты класса *Discrete* описывают дискретные переменные. Для поиска и исправления опечаток была разработана структура словаря.

Словарь состоит из ключей и значений. Одному значению может соответствовать множество ключей.

Поля объекта *Discrete*, `key` (ключ) и `value` (значение), являются переменными R-типа `list`, т.е. они имеют вложенную структуру, представляющую из себя массив массивов, и могут содержать в себе сочетания любых типов данных. Это позволяет эффективно, т.е. в одном объекте, хранить

разнородную информацию. Более подробное описание работы со структурой словаря описано ниже, в пункте 2.2.4.1.

Задать новые значения для полей класса *Discrete*, *key* и *value*, можно при помощи созданных методов *SetValue()* и *SetKey()*, а получить их текущие значения при помощи *GetValue()* и *GetKey()* соответственно. Эти методы также доступны в дочернем классе *Binary*.

Дочерний класс *Binary* имеет более узкую специализацию и описывает только бинарные категориальные переменные (например, пол). Во время инициализации класса *Binary* значения поля *value* устанавливаются по умолчанию: 0 и 1. При необходимости, они могут быть изменены при помощи родительского метода *SetValue()*.

Объекты класса *Dates* необходимы для описания столбцов таблицы, содержащих даты.

Метод *FindErrors()* для классов *Discrete* и *Dates* выполняет поиск опечаток, но для каждого из этих двух классов существует своя реализация данного метода, более подробно это описано в пункте 2.2.4

2.2.3 Типы ошибок

Процесс валидации «сырых» данных должен выявлять следующие типы ошибок:

- Опечатки.
- Неупорядоченные даты.
- Выбросы.
- Пропущенные значения (незаполненные поля).

Для каждого типа ошибок был создан свой класс, который наследуется от класса *Error*. Это сделано для объединения их по общим признакам, таким как:

- Индексы ошибок в таблице.
- Стилль типа ошибки для раскраски итоговой таблицы.
- Название типа ошибки в легенде таблицы.

- Позиция условного обозначения типа ошибки внутри легенды таблицы.

На рисунке 5 изображена структура классов «Типы ошибок».

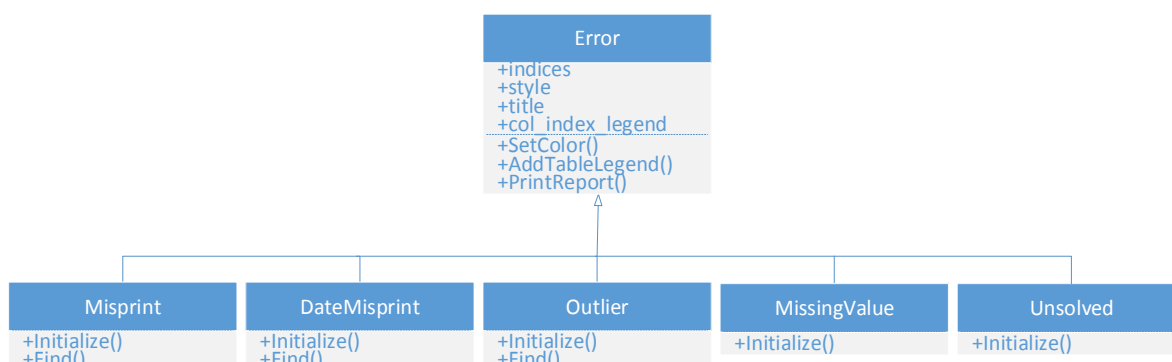


Рисунок 5 – Структура классов «Типы ошибок»

Родительский класс *Error* имеет следующие поля: *indices*, *style*, *title* и *col_index_legend*. Поле *indices* содержит индексы ячеек с ошибками в исходной таблице, поле *style* используется для хранения названия стиля, применяемого в раскраске неправильно заполненных ячеек, поле *title* содержит название типа ошибки (используется для легенды итоговой таблицы), а поле *col_index_legend* хранит позиция ячейки внутри легенды таблицы.

Значения для полей *style*, *title* и *col_index_legend* устанавливаются во время инициализации каждого из дочерних классов *Error*. Для этого используется созданный метод *Initialize()*, а не стандартный конструктор.

Каждый вызов метода *FindErrors()* изменяет поле *indices*, добавляя новые индексы ячеек таблицы, содержащих ошибки.

После того, как был выполнен анализ всех столбцов таблицы, итоговая таблица может быть раскрашена. Это реализует метод *SetColor()*, который взаимодействует с объектом новой рабочей книги Excel, созданной при вызове метода *CreateExcelWB()*. В методе *SetColor()* осуществляется доступ ко всем ячейкам рабочей книги и устанавливаются новые стили для тех ячеек, индексы которых содержит поле *indices*. Выбор стиля для типа ошибки осуществляется исходя из значения, содержащегося в поле *style*. Внутри метода *SetColor()*

также осуществляется вызов метода *AddTableLegend()*, при помощи которого происходит добавление условного обозначения в легенду таблицы.

Метод *PrintReport()* отвечает за запись сообщений о найденных ошибках в пользовательский текстовый файл. Он принимает в качестве одного из аргументов объект класса *Report* и использует его поле *file* с открытым соединением. Тип ошибки определяется автоматически, путем получения значения поля *title* переданного методу объекта. Фрагмент отчета об ошибках представлен на рисунке 6.

```
исправление в строке 5 столбце 24 (Дата вмешательства) с '13.05.2009' на '13.05.2009'
исправление в строке 6 столбце 24 (Дата вмешательства) с '12.05.2009' на '12.05.2009'
исправление в строке 7 столбце 24 (Дата вмешательства) с '18.08.2009' на '18.08.2009'
Опечатка в строке 122 столбце 25 (Росса) значение ячейки = 'один'
Опечатка в строке 123 столбце 25 (Росса) значение ячейки = 'один'
Опечатка в строке 124 столбце 25 (Росса) значение ячейки = 'один'
Опечатка в строке 125 столбце 25 (Росса) значение ячейки = 'один'
Пропущенное значение в строке 29 столбце 26 (Время ИК)
Пропущенное значение в строке 89 столбце 26 (Время ИК)
Пропущенное значение в строке 90 столбце 26 (Время ИК)
Выброс в строке 4 столбце 26 (Время ИК) значение ячейки = '1434'
Выброс в строке 5 столбце 26 (Время ИК) значение ячейки = '1250'
Выброс в строке 6 столбце 26 (Время ИК) значение ячейки = '1410'
Выброс в строке 10 столбце 26 (Время ИК) значение ячейки = '483'
Выброс в строке 35 столбце 26 (Время ИК) значение ячейки = '443'
Выброс в строке 45 столбце 26 (Время ИК) значение ячейки = '592'
Выброс в строке 70 столбце 26 (Время ИК) значение ячейки = '480'
Выброс в строке 97 столбце 26 (Время ИК) значение ячейки = '467'
Выброс в строке 104 столбце 26 (Время ИК) значение ячейки = '436'
Выброс в строке 125 столбце 26 (Время ИК) значение ячейки = '526'
```

Рисунок 6 – Фрагмент отчета об ошибках

2.2.4 Описание реализации поиска опечаток

Поиск опечаток и их исправление, осуществляется при помощи метода *Find()*. У него существует три реализации для каждого из четырех типов значений исследуемой таблицы, описанных выше. Листинг программного кода приведен в приложении А.

2.2.4.1 Метод поиска опечаток для дискретных значений

Доступ к элементам определенного столбца в исходной таблице данных осуществляется по его индексу, который был передан при вызове метода. Полученные значения элементов присваиваются новой переменной, представляющей собой одномерный массив. В цикле, последовательно проверяется каждый элемент этого массива на наличие различных ошибок.

В первую очередь осуществляется поиск пропущенных значений. Если проверка выявила существование пропущенного значения, то его индексы

этого элемента передается полю объекта класса *missingValue* для дальнейшей раскраски, и вызывается метод *PrintReport()*, который производит запись сообщения об ошибке в пользовательский текстовый файл-отчет.

Если результат проверки на заполнение оказался успешным и значение элемента отлично от пустой строки, то начинается проверка на совпадение с одним из значений словаря. Для корректности сравнения, значение словаря и значение элемента столбца приводятся к верхнему регистру.

Если значение элемента не совпало ни с одним значение словаря, проверяется совпадение элемента с одним из ключей. Оба значения так же приводятся к верхнему регистру.

Если совпадение найдено, то по нужному индексу в итоговой таблице производится замена ключа на соответствующее ему значение в словаре, а индексы элемента передаются полю объекта класса *Misprint* для раскраски итоговой таблицы. Далее происходит вызов метода *PrintReport()*. Ячейка, содержащая данную ошибку, в итоговой таблице будет помечена как исправленная опечатка.

В случае, когда элемент столбца не совпал ни с одним из ключей, его индексы в таблице (номер строки и номер столбца) передаются полю объекта класса *Unsolved*, и производится запись сообщения об ошибке в файл при помощи метода *PrintReport()*.

Наглядное представление описанного алгоритма приведено на рисунке 7 в виде схемы.

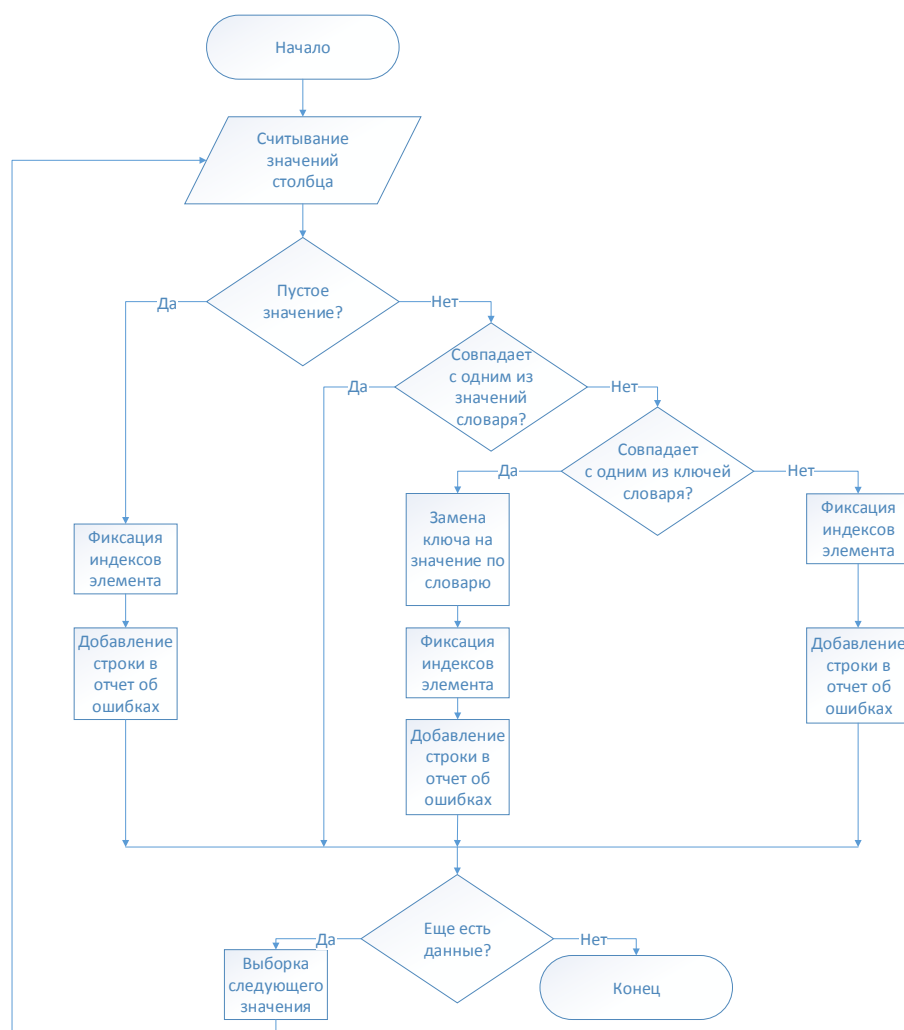


Рисунок 7 – Схема работы алгоритма поиска опечаток для дискретных значений

2.2.4.2 Метод поиска опечаток для непрерывных значений

Так же, как и в методе, реализованном для категориальных переменных, в первую очередь, после получения значений определенного столбца, производится их проверка на заполнение.

Далее значение каждого элемента сравнивается с шаблоном, заданным регулярным выражением, изображенным ниже.

"^((\\d)+)[.]|([[:space:]])?(\\d)+)?\$"

где

^ – символ начала строки,

((\\d)+) – одно число и более,

[.] – один из двух указанных символов (точка или запятая),

| – логическое ИЛИ,

([[:space:]])? – пустые символы ноль или один раз,

\$ – символ конца строки.

Данный шаблон был разработан для работы с целыми и дробными числами.

Если значение элемента не соответствует шаблону, это может означать, что оно содержит буквы, или любые другие символы. Следовательно, индексы элемента передаются полю объекта класса *Unsolved*, который, как было сказано ранее, хранит индексы элементов, которые будут помечены в таблице как неисправленные опечатки. Далее производится запись сообщения о найденной ошибке в файл при помощи метода *PrintReport()*.

Если значение элемента соответствует шаблону, то производится дополнительная проверка разделителя в записи десятичной дроби и поиск лишних пробелов. Если разделитель неверный, то производится его замена, а если найден лишний пробел – он удаляется, а индексы элемента передаются полю объекта класса *Misprint* для раскраски итоговой таблицы. Метод *PrintReport()* сообщает пользователю о найденной и исправленной опечатке, если таковая выявлена. Наглядное представление описанного алгоритма приведено на рисунке 8 в виде схемы.

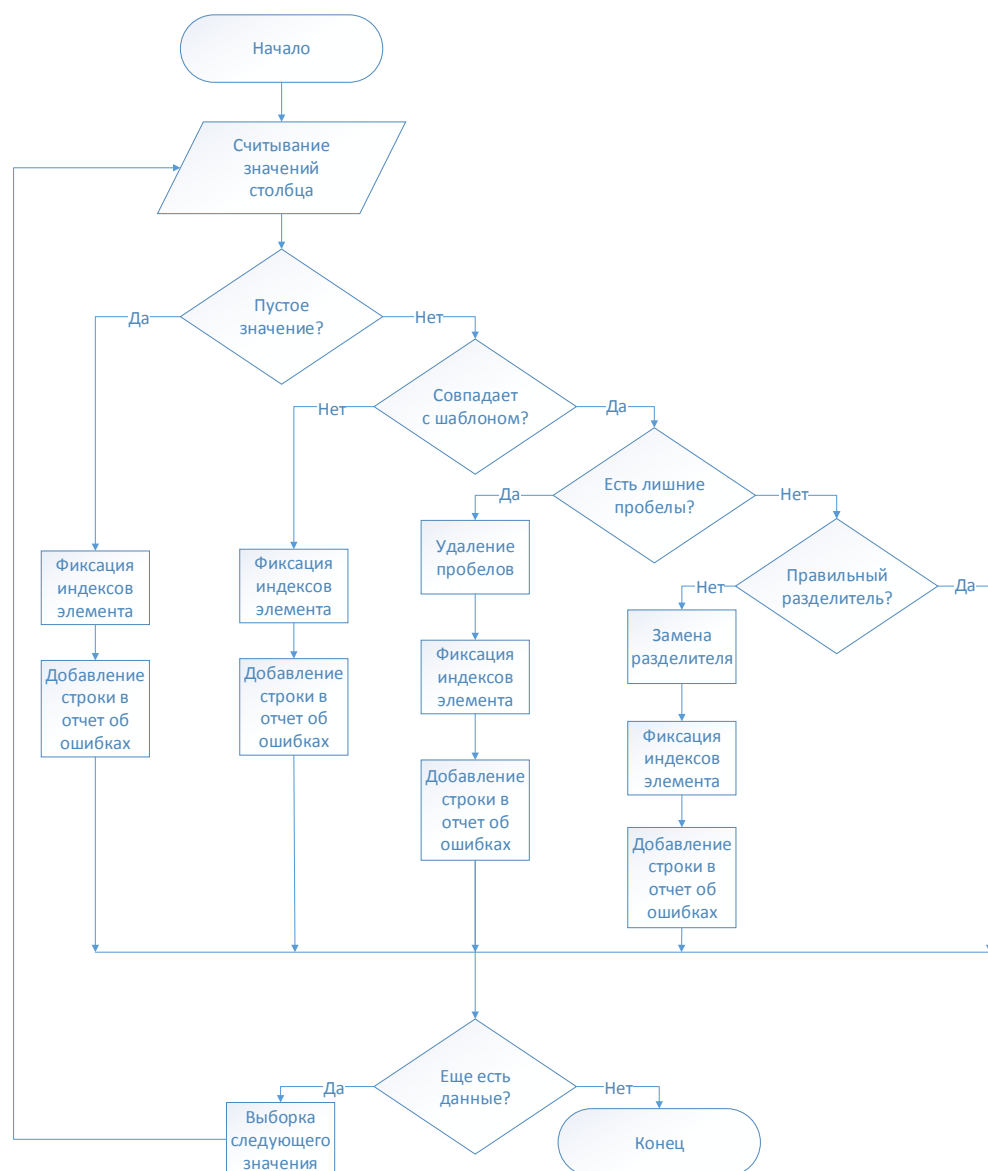


Рисунок 8 – Схема работы алгоритма поиска опечаток для непрерывных значений

2.2.4.3 Метод поиска опечаток для дат

Данная реализация схожа с предыдущей, но отличается применяемым шаблоном регулярного выражения, который используется для сравнения со значением элемента. Шаблон регулярного выражения приведен ниже.

"^((\\d){2})([.][-/])(\\d{2})([.][-/])(\\d{2}|\\d{4})\$",

где

^ – символ начала строки,

((\\d){2}) – два числа,

([.][-/]) – один из четырех указанных символов («,», или «.» , или «/», или «/» или «-»),

((\\d){2}|\\d{4}) – два или четыре числа,

\$ – символ конца строки.

Если значение элемента не соответствует шаблону, то производится запись сообщения об ошибке в пользовательский файл, а индексы этого элемента передаются полю объекта класса *Unsolved*.

Все значения столбца, которые подошли под указанный выше шаблон, являются датами, но существует возможность того, что они записаны неверно. Поэтому производится дополнительная проверка разделителей между числами в записи даты и замена их на точку, если разделитель неправильный. В случае ошибки индексы элемента передаются полю объекта класса *Misprint*, а вызванный метод *PrintReport()* сообщает о найденной и исправленной опечатке. Наглядное представление описанного алгоритма приведено на рисунке 9 в виде схемы.

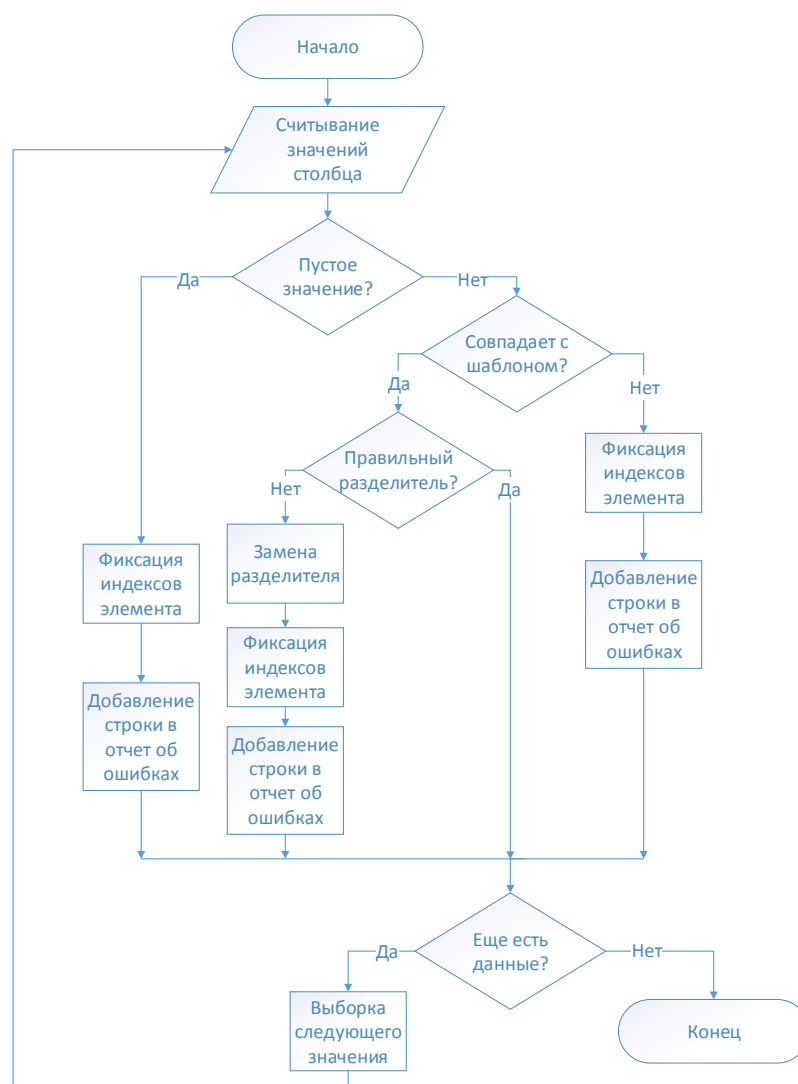


Рисунок 9 – Схема работы алгоритма поиска опечаток для дат

2.2.5 Реализация поиска выбросов

Поиск выбросов осуществляется только среди значений элементов столбцов, которые описывает класс *Continuous*, т.е. непрерывных переменных. Анализ значений возможен лишь в том случае, когда среди них нет неисправленных опечаток. В противном случае результаты анализа будут ошибочными и, следовательно, бесполезными.

Выбросы в столбце определяются при помощи функции *boxplot.stats()*, производной графической функции высокого уровня *boxplot()*, которая служит для построения диаграмм размахов. *boxplot.stats()*, в свою очередь, используется для сбора статистики при создании этих диаграмм.

Диаграммы размахов, или «ящики с усами» (англ. box-whisker plots), получили свое название за характерный вид. Точку или линию, соответствующую медиане, окружает прямоугольник («ящик»), длина которого соответствует одному из показателей разброса или точности оценки генерального параметра. Дополнительно от этого прямоугольника отходят «усы», также соответствующие по длине одному из показателей разброса или точности. Строение получаемых при помощи этой функции «ящичков с усами» представлено ниже на рисунке 10.

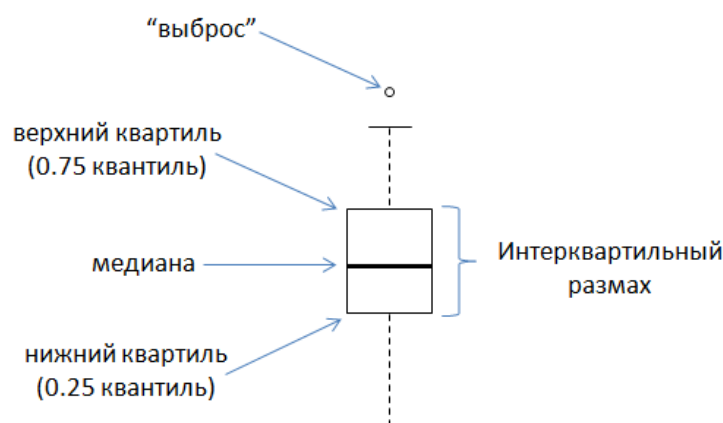


Рисунок 10 – Диаграмма размахов

В R при построении диаграмм размахов используются устойчивые оценки центральной тенденции (медиана) и разброса (интерквартильный размах, далее ИКР).

Медианой случайной величины является такое число, что вероятность получить значение случайной величины справа от него равна вероятности получить значение слева от него (обе они равны $1/2$).

Квантиль в математической статистике – это значение, которое заданная случайная величина не превышает с фиксированной вероятностью [10]. Квантиль порядка α – числовая характеристика закона распределения случайной величины; такое число, что данная случайная величина попадает левее его с вероятностью, не превосходящей α .

Термин «квартиль» используют для обозначения квантиля порядка α , когда α кратно $1/4$. Выделяют нижний квартиль при $\alpha = 1/4$ и верхний квартиль

при $\alpha = 3/4$. Квартиль, соответствующий $\alpha = 1/2$, имеет собственное название – медиана.

Интерквартильным размахом называется разность между третьим и первым квартилями. Простейшие способы определения выбросов основаны на интерквартильном расстоянии. Например, всё, что не попадает в диапазон, представленный формулой ниже, считается выбросами.

$$[Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)],$$

где

Q_1 – первый квартиль,

Q_3 – третий квартиль,

k – неотрицательная постоянная Тьюки, равная 1,5.

Верхний «ус» простирается от верхней границы «ящика» до наибольшего выборочного значения, находящегося в пределах заданного расстояния равного произведению коэффициента 1,5 на ИКР от этой границы. Нижний «ус» строится аналогично верхнему от нижней границы «ящика» до наименьшего выборочного значения. Наблюдения, находящиеся за пределами «усов», потенциально могут быть выбросами.

Далее описана программная реализация поиска выбросов.

В первую очередь выполняется проверка значения счетчика неисправленных опечаток. Если он равен нулю, то это означает, что можно производить дальнейший анализ значений элементов столбца. В противном случае, происходит вызов метода печати в файл *PrintReport()* и запись предупреждения о невозможности проведения дальнейшего анализа, т.к. среди значений столбца присутствуют неисправленные опечатки.

Далее осуществляется доступ к элементам столбца и поиск среди них пропущенных значений.

Функция *boxplot.stats()* работает только с числовыми значениями. Опечатки в столбце, содержащем непрерывные переменные, препятствуют правильному распознаванию его значений средой R как числовых, вместо этого

они представляются как значения строкового типа. Поэтому, для того, чтобы включить эти значения в анализ выбросов, используется шаблон, изображенный ниже, заданный при помощи регулярного выражения.

$$"^\backslash d + ([.,]\backslash d +)? \$",$$

где

$^$ – символ начала строки,

$\backslash d +$ – одно число и более,

$[.,]$ – один из двух указанных символов (точка или запятая),

$([.,]\backslash d +)?$ – разделитель, одно число и более ноль или один раз,

$\$$ – символ конца строки.

Все значения элементов столбца сравниваются с данным шаблоном и при совпадении сохраняются в новую буферную переменную, являющуюся одномерным массивом. Далее, среди значений этой переменной производится замена разделителей в десятичных дробях на точку. После замены используется приведение типа каждого элемента переменной к числовому, и выполняется поиск выбросов с помощью функции *boxplot.stats()*, которая анализирует сразу весь массив элементов. Результат работы функции хранится в виде вложенной структуры R-типа *list*. Значения (не их индексы) из переданного массива, потенциально являющимися выбросами, содержатся в элементе *out* данной структуры. Если он содержит какие-либо значения, то они сопоставляются с исходными значениями столбца, для того, чтобы определить позицию уже найденных выбросов в таблице. Для сравнения разделители в исходных элементах, все еще являющихся строковыми, тоже преобразуются. Во время вызова метода *PrintReport()* происходит печать сообщения о найденных выбросах. Листинг программного кода приведен в приложении А.

2.2.6 Реализация поиска неупорядоченных дат

Метод *Find()* для класса *DateMisprint* реализует поиск непоследовательных дат внутри таблицы. Двумя из его аргументов являются

два объекта класса *Dates*, которые описывают столбцы с разными датами, которые предстоит проверить. Например, это могут быть даты первичных и повторных замеров или даты поступления и выписки пациента. После сортировки столбцов по индексам производится приведение каждого из их элементов к типу данных *R, date*, с форматом представления месяц.день.год. После приведения типов производится присваивание полученных значений двум новым переменным. Далее эти переменные поэлементно сравниваются между собой. Если какое-либо из значений элементов первого столбца дат оказалось больше значения второго столбца дат в этой же строке, то индексы такого элемента добавляются в поле *indices* объекта *DateMisprint*. Далее происходит вызов метода *PrintReport()*, с помощью которого добавляется новое сообщение об ошибке в текстовый файл. В результирующей таблице будут выделены ячейки обоих столбцов в этой строке. Листинг программного кода приведен в приложении А.

2.2.7 Создание сводной таблицы значений

Данный функционал необходим для того, чтобы перед началом проверки «сырых» данных таблицы на валидность, пользователь получил представление о значениях, которые она содержит. Исходя из этого, он сможет верно сопоставить столбцы таблицы (вернее, их индексы) и классы, которые должны их описывать, а так же задать ключи и значения словарей, которые будут использоваться для поиска и исправления опечаток.

Из приведенного на рисунке 11 фрагмента сводной таблицы видно, что в столбце «пол» исходной таблицы содержатся значения: «0», «1», «Ж», «жен», «женщина», «М», «муж», «мужчина». Частота встречаемости каждого из этих значений указана на следующей строке сводной таблицы. Это можно рассматривать как гистограмму, изображенную на рисунке 12. Столбцы гистограммы – это различные значения, а их высота – частота встречаемости. Исходя из этих данных, можно определить словарь для столбца «пол». Ключи: «М», «муж», «мужчина» и «Ж», «жен», «женщина». И соответствующие этим

ключам допустимые значения: «1» и «0». Данный словарь будет использоваться для поиска и замены опечаток. Результат обработки столбца «пол» будет содержать только значения «1» и «0».

	1	2	3	4	5	6	7	8	9	10
1	Пациент									
2	CABG	PCI								
3	51	72								
4	пол									
5	0	1	Ж	жен	женщина	М	муж	мужчина		
6	12	60	9	1	1	38	1	1		
7	возраст									
8	22	24	25	32	37	38	40	43	44	45
9	1	1	1	2	2	2	1	1	2	1
10	вес									
11	45	53	55	56	57	58	59	60	60.5	61
12	2	1	2	2	3	1	2	2	1	3
13	диабет									
14	0	1	1 тип	2 тип						
15	113	8	1	1						
16	Класс по NYHA									
17	2	3								
18	28	95								
19	ТИА\ОНМК									
20	0	1								
21	118	5								
22	АГ ст									
23	0	1	2	3						
24	17	17	28	61						

Рисунок 11 – Фрагмент сводной таблицы

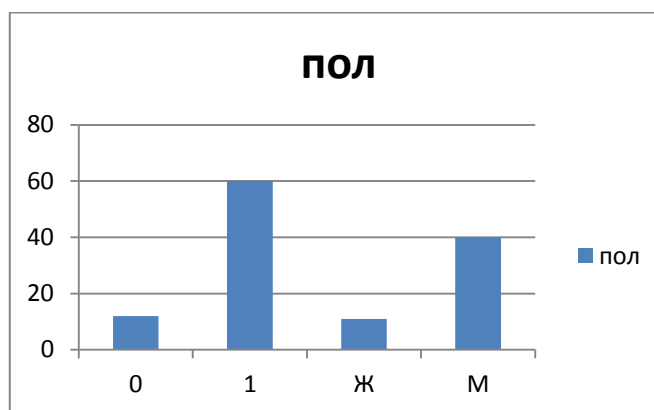


Рисунок 12 – Гистограмма для столбца «пол» исходной таблицы

Реализацией такого функционала является метод *ColumnsValues()*, который работает с объектом класса *SummaryTable*. В нем названия столбцов исходной таблицы, полученные при помощи функции *colnames()*, присваиваются новой переменной. Далее, в цикле к элементам каждого столбца исходной таблицы применяется функция *table()*. Ее результат содержит все различные значения столбца и частоту встречаемости каждого из этих значений. Производится преобразование типа результата функции *table()* в R-

тип таблицу данных (`data.frame`). Для того чтобы получить нужный формат используется функция транспонирования таблицы данных как матрицы, а после используется обратное приведение типа. Полученная таблица добавляется в созданную ранее пустую сводную таблицу данных в виде новых строк. Т.к. количество различных значений в разных столбцах таблицы неодинаковое, это создает проблемы для создания сводной таблицы обычным путем – ее строки должны быть одной длины. Поэтому для решения этой проблемы используется функция `rbind.fill()` из библиотеки `plyr`, которая заменяет значения недостающих столбцов на NA. Если количество различных значений в одном столбце больше указанного пользователем, например, десять, то для отображения будут использованы только первые десять значений столбца.

Названия столбцов исходной таблицы вставляются перед каждой новой строкой, описывающей различные значения и их частоту. Далее происходит запись полученной сводной таблицы в новую рабочую книгу Excel, путем вызова родительского метода `CreateExcelWB()`. Устанавливаются необходимые стили оформления и при помощи метода `SaveExcelWB()` производится сохранение нового Excel-файла.

3 РАЗРАБОТКА БИБЛИОТЕКИ

После написания классов и методов, все содержащие их файлы были объединены в новую библиотеку, руководствуясь книгой Хэдли Уикхэма [11]. Библиотеки являются основными единицами воспроизводимого кода R. Они включают многократно используемые функции R, документацию, которая описывает, и примеры данных. Для создания новой библиотеки под названием «exploration», использовались вспомогательные библиотеки devtools и roxygen2.

С помощью библиотеки devtools был создан каталог новой библиотеки, а так же автоматически были созданы файлы DESCRIPTION и NAMESPACE.

Далее, все файлы, в которых определены необходимые для работы классы и методы, были скопированы в созданный каталог.

После этого, к библиотеке была добавлена документация. Это сделано для того, чтобы другие пользователи понимали, как использовать определенные классы и методы. Для этого использовалась вспомогательная библиотека roxygen2, которая позволяет создавать документацию в стандартном для R формате. Способ написания документации заключается в добавлении специальных комментариев к началу каждой каждого скопированного файла, которые позже будут скомпилированы в правильный формат документации пакета. Пример документации приведен на рисунке 13. В процессе написания документации изменяется содержимое файла DESCRIPTION.

Задача файла DESCRIPTION – хранить важные метаданные о библиотеке. Например, записи того, какие другие библиотеки и их версии необходимы для запуска новой библиотеки. Также он указывает основную информацию о библиотеке (для чего она была создана), кто может ее использовать (лицензия) и всю контактную информацию разработчика для обратной связи, в случае если возникнут какие-либо проблемы. Еще одной важной частью файла DESCRIPTION является описание списка имен R-файлов, которые должны быть загружены до текущего файла, т.к. код S4 часто должен выполняться в определенном порядке.

Для лучшего взаимодействия с другими библиотеками, в новой библиотеке должно быть определено, какие функции она предоставляет другим библиотекам, и какие функции она требует от других библиотек. Это описывается в файле `NAMESPACE`. Изменение файла `NAMESPACE` было произведено с использованием библиотеки `roxygen2`.

После выполнения описанных выше шагов была выполнена стандартная установка новой библиотеки.

Конечным этапом было размещение новой библиотеки на репозитории GitHub. Преимущество заключается в том, что установку и обновление новой библиотеки можно производить непосредственно с GitHub.

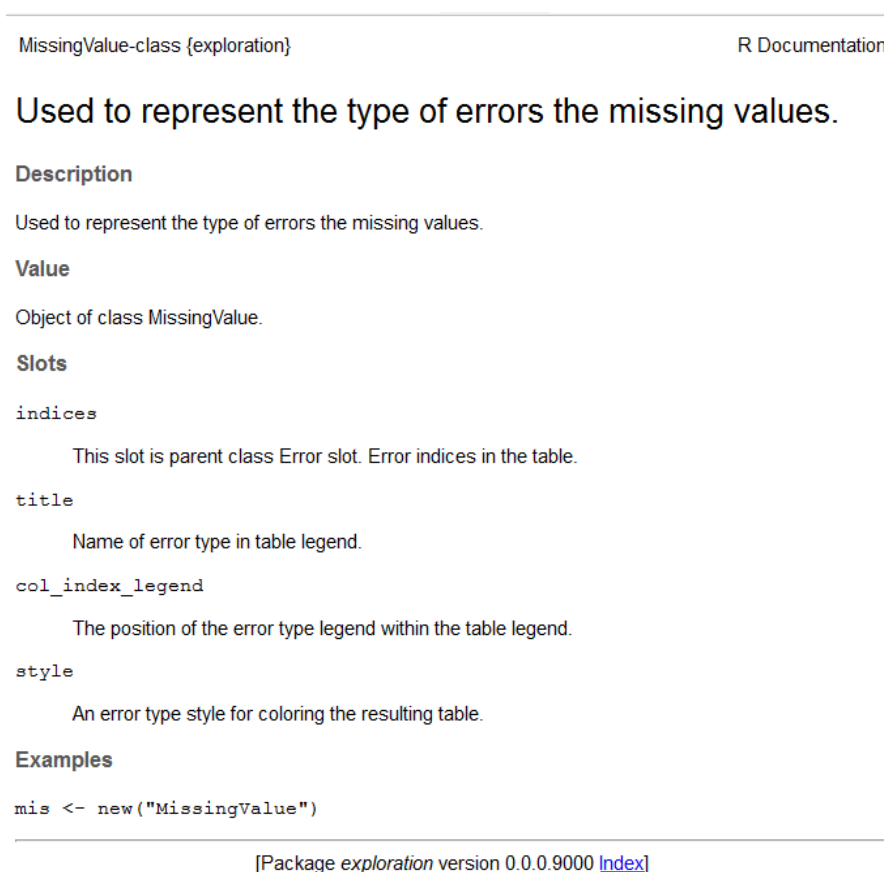


Рисунок 13 – Страница из документации библиотеки

4 ОТЛАДКА И ТЕСТИРОВАНИЕ

На этапе отладки и тестирования на реальных данных было выявлено и исправлено несколько ошибок.

Текстовый файл-отчет, сводная таблица и файл с итоговой раскрашенной таблицей каждый раз перезаписывались, было невозможно посмотреть их ранние версии. Эта проблема была решена путем добавления текущего времени и даты в названия файлов.

Среда R не всегда верно интерпретирует значения в столбцах таблицы. Из-за опечаток, например, лишнего пробела после запятой в десятичной дроби, все числовые значения столбца становятся строковыми и не включаются в анализ наряду с числовыми. Данная проблема была исправлена при помощи шаблонов регулярных выражений и приведения типов.

Во время тестирования метода для создания сводной таблицы, которая содержит значения каждого столбца, и частоту встречаемости этих значений, была выявлена ошибка. Длины столбцов получаются разными, а для компоновки таблицы данных или матрицы они должны быть одной длины. Ошибка была устранена с помощью функции *rbind.fill()* из библиотеки *plyr*, которая заменяет значения недостающих столбцов на NA.

5 РЕЗУЛЬТАТЫ ОБРАБОТКИ

Разработанные алгоритмы были проверены на реальных входных данных. Далее на рисунках будут представлены входные данные, содержащие ошибки и результат их обработки.

На рисунке 14 показан фрагмент входных данных, содержащих пропущенные значения. На рисунке 15 показан фрагмент итоговой таблицы с найденными и выделенными незаполненными ячейками.

	23	24
121		09.07.2013
122		09.04.2013
123		29.05.2012
124		25.02.2012

Рисунок 14 – Фрагмент данных с пропущенными значениями

	23	24
121		24.09.2013
122		09.07.2013
123		09.04.2013
124		29.05.2012
125		25.02.2012

Рисунок 15 – Результат обработки входных данных

На рисунке 16 изображен фрагмент входных данных, который содержит нарушение упорядоченности дат, а на рисунке 17 представлена итоговая раскрашенная таблица с выделением этой ошибки.

24	42
Дата вмешательства	
19.02.2009	20.02.2008
08.05.2009	09.05.2009

Рисунок 16 – Фрагмент данных с неупорядоченными датами

24	42
Дата вмешательства	
19.02.2009	20.02.2008
08.05.2009	09.05.2009

Рисунок 17 – Фрагмент исправленной таблицы

Опечатки в виде данных, не входящих в набор определенных допустимых значений отображены на рисунке 18. Исправление этих опечаток показано на рисунке 19.

1	2
CABG	M
CABG	M
CABG	M
CABG	Ж
CABG	M
CABG	M
CABG	M
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	0

Рисунок 18 – Фрагмент входных данных с опечатками

1	2
CABG	1
CABG	1
CABG	1
CABG	1
CABG	0
CABG	1
CABG	1
CABG	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	1
PCI	0

Рисунок 19 – Исправленные опечатки

Наличие такой опечатки, как лишний пробел в записи десятичных дробей, хорошо видно на рисунке 20. На рисунке 21 представлен фрагмент таблицы, в которой данная ошибка исправлена.

16	17
0,88	1
0, 92	1
1,1	2
1,3	1

Рисунок 20 – Лишний пробел в исходных данных

16	17
0,78	2
0,88	1
0,92	1
1,1	2
1,3	1

Рисунок 21 – Фрагмент таблицы с удаленным лишним пробелом

Неправильные разделители между числами в записи дат показаны на рисунке 22. В результате обработки они были найдены и заменены на точки, это показано на рисунке 23.

24	25
13,05,2009	1
12,05,2009	1
18,08,2009	1
14.10.2009	1

Рисунок 22 – Неправильные разделители в записи дат

24	25
08.05.2009	1
13.05.2009	1
12.05.2009	1
18.08.2009	1
14.10.2009	1

Рисунок 23 – Исправленные разделители

На рисунке 24 отображен фрагмент входных данных, а на рисунке 25 выделены значения из этого же фрагмента, потенциально являющиеся выбросами.

26	27
Время ИК	Градиент
197	5
1434	11
1250	4
1410	12
176	12
334	16
173	6
483	13
231	13

Рисунок 24 – Фрагмент входных данных

26	27
1434	11
1250	4
1410	12
176	12
334	16
173	6
483	13
231	13

Рисунок 25 – Потенциальные выбросы

Проверка нормальности распределения проводится после того, как все найденные во входных данных ошибки исправлены, в том числе и выбросы. На рисунке 26 приведены результаты проверки нормальности распределения данных пациентов из первой группы, а на рисунке 27 – пациентов из второй группы.

1	2	3	4	5	6	7
	n	shapiro, p-level	ad, p-level	cvm, p-level	lillie, p-level	sf, p-level
возраст	52	0,015	0,066	0,082	0,082	0,01
вес	52	0,804	0,859	0,896	0,896	0,754
euroscore.5.5.10..10	52	0	0	0	0	0
STS.score..3.3.8..8	52	0	0	0	0	0
Градиент.давления.пиковый	52	0,024	0,02	0,024	0,024	0,025
градиент.давления.средний	52	0,003	0,003	0,005	0,005	0,003
S.АоКл	52	0,024	0,053	0,067	0,067	0,027
ФВ	52	0	0	0	0	0
Время.ИК	51	0	0	0	0	0
Градиент.давления.пиковый.1	51	0,004	0,001	0,001	0,001	0,012
градиент.давления.средний.1	50	0,168	0,054	0,048	0,048	0,171

Рисунок 26 – Результаты проверки нормальности распределения данных пациентов из первой группы

1	2	3	4	5	6	7
	n	shapiro, p-level	ad, p-level	cvm, p-level	lillie, p-level	sf, p-level
возраст	53	0,001	0	0	0	0,001
вес	53	0,279	0,297	0,288	0,288	0,225
euroscore.5.5.10..10	53	0	0	0	0	0
STS.score..3.3.8..8	53	0	0	0	0	0
Градиент.давления.пиковый	53	0,013	0,037	0,036	0,036	0,014
градиент.давления.средний	53	0,001	0,001	0,003	0,003	0,004
S.АоКл	53	0,015	0,003	0,002	0,002	0,03
ФВ	53	0,002	0,005	0,017	0,017	0,003
Время.ИК	51	0	0	0	0	0
Градиент.давления.пиковый.1	52	0	0	0	0	0
градиент.давления.средний.1	20	0,712	0,515	0,572	0,572	0,45

Рисунок 27 – Результаты проверки нормальности распределения данных пациентов из второй группы

На рисунке 28 приведен график плотности распределения для столбца «Вес» из исправленных данных.

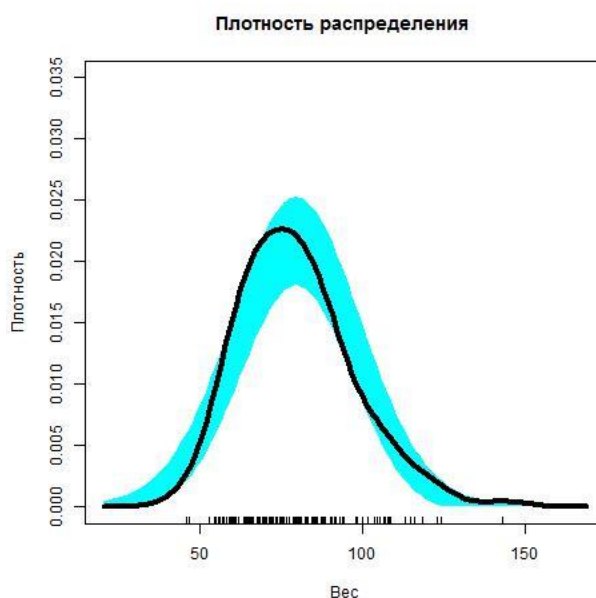


Рисунок 28 – График плотности распределения для столбца «Вес»

Фрагмент сводной таблицы показан на рисунке 29. Она содержит названия столбцов исходной таблицы, различные значения каждого столбца и частоту встречаемости этих значений.

	A	B	C	D	E	F	G	H	I	J
1	Пациент									
2	CABG	PCI								
3	51	72								
4	пол									
5	0	1	Ж	М						
6	12	60	11	40						
7	возраст									
8	22	24	25	32	37	38	40	43	44	45
9	1	1	1	2	2	2	1	1	2	1
10	вес									
11	45	53	55	56	57	58	59	60	60.5	61
12	2	1	2	2	3	1	2	2	1	3
13	диабет									
14	0	1	1 тип	2 тип						
15	113	8	1	1						
16	Класс по NYHA									
17	2	3								
18	28	95								
19	ТИА\ОНМК									
20	0	1								
21	118	5								
22	АГ ст									
23	0	1	2	3						
24	17	17	28	61						

Рисунок 29 – Фрагмент сводной таблицы

ЗАКЛЮЧЕНИЕ

Разработанная в рамках дипломной работы библиотека, решает поставленные задачи проверки входных данных, а также позволяет биостатистикам анализировать данные исследования в удобной форме за короткий промежуток времени.

Созданная библиотека алгоритмов для статистического анализа данных клинических исследований удовлетворяет всем поставленным требованиям:

- Выявление пропущенных значений (незаполненных полей).
- Поиск опечаток.
- Поиск выбросов.
- Исследование нормальности распределения различными статистическими критериями.
- Проверка на упорядочение дат.

В ходе выполнения дипломной работы была изучена предметная область, разработана и описана архитектура библиотеки, идентифицирующей потенциальные проблемы исследования данных, используя ООП модель S4 языка R. Была изучена и использована система контроля версий Git, получены навыки работы с репозиторием GitHub, ветвлением, устранением конфликтов. Было произведено тестирование библиотеки на реальных задачах.

Таким образом, удалось создать библиотеку проверки входных данных, которая в дальнейшем позволит переложить рутинные действия на компьютер и даст возможность биостатистикам анализировать данные исследования удобной форме.

Выражаю искреннюю благодарность моему научному руководителю, Лукинову Виталию Леонидовичу, за помощь в подготовке данной работы и поддержку.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Румянцев П.О. Статистические методы анализа в клинической практике [Электронный ресурс] / П.О. Румянцев, С.Ю. Чекин, У.В. Румянцева, В.А. Саенко. — Электрон. ст. — М., 2009, — URL: <https://elibrary.ru/item.asp?id=13081120>, свободный. — Яз. рус. — Аналог печат. изд. (Проблемы эндокринологии. — 2009. — № 5. — С.48-55). — (Дата обращ. 27.09.2017).
- 2 Мастицкий С.Э. Статистический анализ и визуализация данных с помощью R [Электронный ресурс] / С.Э. Мастицкий, Шитиков В.К.; –Хайдельберг – Лондон – Тольятти., 2014. / – URL: <http://www.ievbras.ru/ecostat/Kiril/R/Mastitsky%20and%20Shitikov%202014.Pdf> /, свободный. — Яз. рус. — (Дата обращ. 27.09.2017).
- 3 Виды клинических исследований лекарственных средств [Электронный ресурс] / – URL:<http://www.medtran.ru/rus/trials/clinicaltrials.htm> / свободный. — Яз. рус. — (Дата обращ. 27.09.2017).
- 4 Нормальное распределение [Электронный ресурс] / – URL: https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5 / свободный. — Яз. рус. — (Дата обращ. 27.09.2017).
- 5 Математическое ожидание [Электронный ресурс] / – URL: https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5 / свободный. — Яз. рус. — (Дата обращ. 27.09.2017).
- 6 Дисперсия случайной величины [Электронный ресурс] / – URL: <http://mathmir.ru/math-articles/chislovye-kharakteristiki-diskretnoi-sluchainoi-velichiny.html> / свободный. — Яз. рус. — (Дата обращ. 27.09.2017).

- 7 Среднеквадратическое отклонение случайной величины [Электронный ресурс] / – URL: <http://mathmir.ru/math-articles/chislovyie-kharakteristiki-diskretnoi-sluchainoi-velichiny.html> / свободный. – Яз. рус. – (Дата обращ. 27.09.2017).
- 8 Платформа по обучению языку R [Электронный ресурс] / – URL: <https://stepik.org/course/497> / свободный. – Яз. рус. – (Дата обращ. 27.09.2017).
- 9 Zuur A.F. A protocol for data exploration to avoid common statistical problems [Электронный ресурс] / A.F. Zuur, E.N. Ieno, C.S. Elphick. — Электрон. ст. — Лондон.: British Ecological Society., 2010, – URL: <http://onlinelibrary.wiley.com/doi/10.1111/j.2041-210X.2009.00001.x/abstract>, свободный. – Яз. англ. — Аналог печат. изд. (Methods in Ecology and Evolution. — 2010. — № 1. — С.3-14). — (Дата обращ. 27.09.2017).
- 10 Квантиль [Электронный ресурс] / – URL: <https://en.wikipedia.org/wiki/Quantile> / свободный. – Яз. англ. – (Дата обращ. 27.09.2017).
- 11 Описание пошагового создания библиотеки языка R [Электронный ресурс] / – URL: <http://r-pkgs.had.co.nz> / свободный. – Яз. англ. – (Дата обращ. 27.09.2017).

ПРИЛОЖЕНИЕ А

Листинг программного кода

```
File<- setClass("File",
slots = c(path = "character")
)

setGeneric(name = "GetPath",
  def = function(theObject)
  {
    standardGeneric("GetPath")
  }
)
setMethod(f = "GetPath",
  signature = "File",
  definition = function(theObject)
  {
    return(theObject@path)
  }
)

setGeneric(name = "SetPath",
  def = function(theObject, new_path)
  {
    standardGeneric("SetPath")
  }
)
setMethod(f = "SetPath",
  signature = "File",
  definition = function(theObject, new_path)
  {
    theObject@path <- new_path
    return(theObject)
  }
)

In <- setClass("In",
  slots = c(table = "data.frame"),
  contains = "File"
)

setGeneric(name = "Read",
  def = function(theObject)
  {
    standardGeneric("Read")
  }
)
setMethod(f = "Read",
  signature = "In",
  definition = function(theObject)
  {
    theObject@table<- read.csv2(theObject@path,
                                na.strings = c("", "NA"),
                                sep = ";",
                                dec = ",",
                                stringsAsFactors = FALSE,
                                check.names = FALSE)

    return(theObject)
  }
)

Out <- setClass("Out",
  slots = c(table = "data.frame",
            sheet_name = "character",
            wb = "jobjRef",
            sheet = "jobjRef",
            row_header = "numeric",
            row_table_legend = "numeric"),
  contains = "File",
  prototype = list(row_header = 1,
                   row_table_legend = 1)
)

setGeneric(name = "setTable",
  def = function(theObject, myfile)
  {
    standardGeneric("setTable")
  }
)
setMethod(f = "setTable",
  signature = "Out",
```

```

    definition = function(theObject, myfile)
    {
      theObject@table <- myfile@table
      return(theObject)
    }
  )

  setGeneric(name = "getExcelSheetName",
    def = function(theObject)
    {
      standardGeneric("getExcelSheetName")
    }
  )

  setMethod(f = "getExcelSheetName",
    signature = "Out",
    definition = function(theObject)
    {
      return(theObject@sheet_name)
    }
  )

  setGeneric(name = "setExcelSheetName",
    def = function(theObject, new_sheet_name)
    {
      standardGeneric("setExcelSheetName")
    }
  )

  setMethod(f = "setExcelSheetName",
    signature = "Out",
    definition = function(theObject, new_sheet_name)
    {
      theObject@sheet_name <- new_sheet_name
      return(theObject)
    }
  )

  setGeneric(name = "CreateExcelWB",
    def = function(theObject, colnames = TRUE, startRow)
    {
      standardGeneric("CreateExcelWB")
    }
  )

  setMethod(f = "CreateExcelWB",
    signature = "Out",
    definition = function(theObject, colnames = TRUE, startRow)
    {
      theObject@wb <- createWorkbook(type = "xlsx")
      theObject@sheet <- createSheet(theObject@wb, theObject@sheet_name)
      createRow(theObject@sheet, rowIndex = 1)
      TABLE_COLNAMES_STYLE <- CellStyle(theObject@wb) +
        Font(theObject@wb, isBold = TRUE) +
        Alignment(wrapText = TRUE, horizontal = "ALIGN_CENTER") +
        Border(position = c("BOTTOM", "LEFT", "TOP", "RIGHT"))

      addDataFrame(theObject@table,
        theObject@sheet,
        row.names = FALSE,
        startRow = startRow,
        startColumn = 1,
        colnamesStyle = TABLE_COLNAMES_STYLE,
        col.names = colnames)
      return(theObject)
    }
  )

  setGeneric(name = "SaveExcelWB",
    def = function(theObject, freeze = FALSE)
    {
      standardGeneric("SaveExcelWB")
    }
  )

  setMethod(f = "SaveExcelWB",
    signature = "Out",
    definition = function(theObject, freeze = FALSE)
    {
      autoSizeColumn(theObject@sheet, colIndex = c(1:ncol(theObject@table)))
      if (freeze) createFreezePane(theObject@sheet, rowSplit = 2, colSplit = 1, startRow = 1,
startColumn = 1)
      saveWorkbook(theObject@wb, theObject@path)
      print("New workbook was created")
      return(theObject)
    }
  )

  Report <- setClass("Report",
    slots = c(file = "file"),
    contains = "File"
  )

```

```

setGeneric(name = "setDirectory",
  def = function(theObject, new_directory)
  {
    standardGeneric("setDirectory")
  }
)
setMethod(f = "setDirectory",
  signature = "Report",
  definition = function(theObject, new_directory)
  {
    theObject@path <- paste(new_directory, "Report_", format(Sys.time(), "%d_%m_%Y_%H_%M_%S"),
".txt", sep = "")
    return(theObject)
  }
)

setGeneric(name = "Create",
  def = function(theObject)
  {
    standardGeneric("Create")
  }
)
setMethod(f = "Create",
  signature = "Report",
  definition = function(theObject)
  {
    theObject@file <- file(description = theObject@path, open = "w")
    return(theObject)
  }
)

setGeneric(name = "close",
  def = function(theObject)
  {
    standardGeneric("close")
  }
)
setMethod(f = "close",
  signature = "Report",
  definition = function(theObject)
  {
    on.exit(close(theObject@file))
  }
)

SummaryTable <- setClass("SummaryTable",
  contains = "out"
)

setGeneric(name = "ColumnsValues",
  def = function(theObject, myfile, only)
  {
    standardGeneric("ColumnsValues")
  }
)
setMethod(f = "ColumnsValues",
  signature = "SummaryTable",
  definition = function(theObject, myfile, only)
  {
    table_in_names <- colnames(myfile@table)
    for (i in 1:ncol(myfile@table))
    {
      tmp <- table(myfile@table[[i]])
      unique_sum <- length(tmp)
      tmp <- as.data.frame(tmp)
      tmp <- t.data.frame(tmp)
      tmp <- as.data.frame.matrix(tmp, stringsAsFactors = FALSE)
      if (unique_sum > only) tmp <- tmp[,1:only]
      theObject@table<- rbind.fill(theObject@table,tmp)
    }

    for(j in 1:length(table_in_names))
    {
      for (i in seq(1, nrow(theObject@table) + length(table_in_names), by = 3))
      {
        theObject@table[seq(i + 1,nrow(theObject@table) + 1), ] <-
theObject@table[seq(i,nrow(theObject@table)), ]
        theObject@table[i,] <- c(table_in_names[j], rep(NA, ncol(theObject@table) - 1))
        j <- j + 1
      }
      break
    }

    theObject <- CreateExcelWB(theObject, colnames = FALSE, startRow = 1)
    ALL_CELLS_STYLE <- CellStyle(theObject@wb) +
      Border(position = c("BOTTOM", "LEFT", "TOP", "RIGHT")) +
      Alignment(wrapText = TRUE, horizontal = "ALIGN_CENTER")
  }
)

```

```

TITLE_STYLE <- ALL_CELLS_STYLE +
  Font(theObject@wb, isBold = TRUE)

allrows <- getRows(theObject@sheet, rowIndex = 1:nrow(theObject@table))
allcells <- getCells(allrows, colIndex = 1:ncol(theObject@table))
title_rows <- allrows[seq(1, length(allrows), 3)]
title_cells <- getCells(title_rows, colIndex = 1:ncol(theObject@table))

for (i in 1:length(allcells))
{
  setCellStyle(allcells[[i]], ALL_CELLS_STYLE)
}

for (i in seq(1, nrow(theObject@table), 3))
{
  addMergedRegion(theObject@sheet, i, i, 1, 10)
}

for(i in 1:length(title_cells))
{
  setCellStyle(title_cells[[i]], TITLE_STYLE)
}

theObject <- SaveExcelWB(theObject)
return(theObject)
}
)

Column <- setClass("Column",
  slots = c(column_index = "numeric")
)

setGeneric(name = "getIndex",
  def = function(theObject)
  {
    standardGeneric("getIndex")
  }
)
setMethod(f = "getIndex",
  signature = "Column",
  definition = function(theObject)
  {
    return(theObject@column_index)
  }
)

setGeneric(name = "setIndex",
  def = function(theObject, index_value)
  {
    standardGeneric("setIndex")
  }
)
setMethod(f = "setIndex",
  signature = "Column",
  definition = function(theObject, index_value)
  {
    theObject@column_index <- index_value
    return(theObject)
  }
)

Continuous <- setClass("Continuous",
  contains = "Column"
)

setGeneric(name = "FindErrors",
  def = function(theObject, myfile_in, myfile_out, myfile_report, misprints, missing_values,
  unsolved_misprints, outliers)
  {
    standardGeneric("FindErrors")
  }
)
setMethod(f = "FindErrors",
  signature = "Continuous",
  definition = function(theObject, myfile_in, myfile_out, myfile_report, misprints, missing_values,
  unsolved_misprints, outliers)
  {
    output_list <- Find(misprints, missing_values, unsolved_misprints, myfile_in, myfile_out,
  myfile_report, theObject)
    outliers <- Find(outliers, output_list$file, myfile_report, theObject,
  output_list$unsolved_number)
    output_list <- c(output_list, "outliers" = outliers)
    return(output_list)
  }
)

Discrete <- setClass("Discrete",
  contains = "Column",
  slots = c(key = "list",

```



```

        value = "list")
    )
    setGeneric(name = "getValue",
      def = function(theObject)
      {
        standardGeneric("getValue")
      }
    )
    setMethod(f = "getValue",
      signature = "Discrete",
      definition = function(theObject)
      {
        return(theObject@value)
      }
    )

    setGeneric(name = "setValue",
      def = function(theObject, newValue, add = FALSE)
      {
        standardGeneric("setValue")
      }
    )
    setMethod(f = "setValue",
      signature = "Discrete",
      definition = function(theObject, newValue)
      {
        theObject@value <- as.list(newValue)
        return(theObject)
      }
    )

    setGeneric(name = "getKey",
      def = function(theObject)
      {
        standardGeneric("getKey")
      }
    )
    setMethod(f = "getKey",
      signature = "Discrete",
      definition = function(theObject)
      {
        return(theObject@key)
      }
    )

    setGeneric(name = "setKey",
      def = function(theObject, newKey, add = FALSE)
      {
        standardGeneric("setKey")
      }
    )
    setMethod(f = "setKey",
      signature = "Discrete",
      definition = function(theObject, newKey)
      {
        if (!is.list(newKey)) newKey <- list(newKey)
        for(i in 1:length(newKey))
        {
          theObject@key[[i]] <- as.list(newKey[[i]])
        }
        return(theObject)
      }
    )

    setGeneric(name = "FindErrors",
      def = function(theObject, myfile_in, myfile_out, myfile_report, misprints, missing_values,
        unsolved_misprints)
      {
        standardGeneric("FindErrors")
      }
    )
    setMethod(f = "FindErrors",
      signature = "Discrete",
      definition = function(theObject, myfile_in, myfile_out, myfile_report, misprints, missing_values,
        unsolved_misprints)
      {
        output_list <- Find(misprints, missing_values, unsolved_misprints, myfile_in, myfile_out,
          myfile_report, theObject)
        return(output_list)
      }
    )

    Binary <- setClass("Binary",
      contains = "Discrete"
    )
    setMethod(f = "initialize",
      signature = "Binary",
      definition = function(.Object)
      {

```

```

        .Object@value[["Zero"]] <- list(0)
        .Object@value[["One"]] <- list(1)
    return(.Object)
}

)

Dates <- setClass("Dates",
    contains = "Column"
)

setGeneric(name = "FindErrors",
    def = function(theObject, myfile_in, myfile_out, myfile_report, misprints, missing_values,
        unsolved_misprints)
    {
        standardGeneric("FindErrors")
    }
)

setMethod(f = "FindErrors",
    signature = "Dates",
    definition = function(theObject, myfile_in, myfile_out, myfile_report, misprints, missing_values,
        unsolved_misprints)
    {
        output_list <- Find(misprints, missing_values, unsolved_misprints, myfile_in, myfile_out,
            myfile_report, theObject)
        return(output_list)
    }
)

Error <- setClass("Error",
    slots = c(indices = "character",
        style = "character",
        title = "character",
        col_index_legend = "numeric")
)

setGeneric(name = "SetColor",
    def = function(theObject, myfile_out)
    {
        standardGeneric("SetColor")
    }
)

setMethod(f = "SetColor",
    signature = "Error",
    definition = function(theObject, myfile_out)
    {
        cat(sprintf("Attention! The painting of the %s is in progress, please wait.",
            tolower(class(theObject)[1])))

        MISSING_VALUE_STYLE <- CellStyle(myfile_out@wb) +
            Font(myfile_out@wb, isItalic = TRUE) +
            Fill(backgroundColor = "gray70") +
            Border(position = c("BOTTOM", "LEFT", "TOP", "RIGHT"))
        MISPRINT_STYLE <- CellStyle(myfile_out@wb) +
            Font(myfile_out@wb, isItalic = TRUE) +
            Fill(backgroundColor = "gold1") +
            Border(position = c("BOTTOM", "LEFT", "TOP", "RIGHT"))
        UNSOLVED_MISPRINT_STYLE <- CellStyle(myfile_out@wb) +
            Font(myfile_out@wb, isItalic = TRUE) +
            Fill(backgroundColor = "darkorange") +
            Border(position = c("BOTTOM", "LEFT", "TOP", "RIGHT"))
        OUTLIERS_STYLE <- CellStyle(myfile_out@wb) +
            Font(myfile_out@wb, isItalic = TRUE) +
            Fill(backgroundColor = "firebrick1") +
            Border(position = c("BOTTOM", "LEFT", "TOP", "RIGHT"))
        DATE_MISPRINT_STYLE <- CellStyle(myfile_out@wb) +
            Font(myfile_out@wb, isItalic = TRUE) +
            Fill(backgroundColor = "lightpink4") +
            Border(position = c("BOTTOM", "LEFT", "TOP", "RIGHT"))

        style <- switch(theObject@style,
            "missing_value" = MISSING_VALUE_STYLE,
            "misprint" = MISPRINT_STYLE,
            "unsolved_misprint" = UNSOLVED_MISPRINT_STYLE,
            "outlier" = OUTLIERS_STYLE,
            "dateMisprint" = DATE_MISPRINT_STYLE)

        rows <- getRows(myfile_out@sheet, rowIndex = 1:nrow(myfile_out@table) + myfile_out@row_header +
            myfile_out@row_table_legend)
        cells <- getCells(rows, colIndex = 1:ncol(myfile_out@table))
        lapply(names(cells[theObject@indices]), function(i) setCellStyle(cells[[i]], style))
        AddTableLegend(theObject, myfile_out, style)
    }
)

setGeneric(name = "AddTableLegend",
    def = function(theObject, myfile_out, style)
    {
        standardGeneric("AddTableLegend")
    }
)

```

```

    }
  )
  setMethod(f = "AddTableLegend",
    signature = "Error",
    definition = function(theObject, myfile_out, style)
    {
      rows <- getRows(myfile_out@sheet, rowIndex = 1)
      sheetTitle <- createCell(rows, theObject@col_index_legend)
      setCellValue(sheetTitle[[1,1]], theObject@title)
      setCellStyle(sheetTitle[[1,1]], style)
    }
  )

  setGeneric(name = "PrintReport",
    def = function(theObject, myfile_report, row_index, col_index=NULL, col_name, value, value_new,
    not_outliers = FALSE)
    {
      standardGeneric("PrintReport")
    }
  )
  setMethod(f = "PrintReport",
    signature = "Error",
    definition = function(theObject, myfile_report, row_index=NULL, col_index, col_name, value,
    value_new, not_outliers = FALSE)
    {
      if (not_outliers == TRUE)
      {
        cat(c("Невозможно определить выбросы, в столбце", col_index, paste0("(", col_name, ")"),
        есть не исправленные опечатки ", "\n"), file = myfile_report@file, append = T)
      } else if (class(theObject)[1] == "Misprint")
      {
        cat(c(theObject@title, "в", "строке", row_index, "столбце", col_index, paste0("(", col_name,
        ")"), ", "c", paste0("(", value, ")", "на", paste0("(", value_new, ")", "\n"), file =
        myfile_report@file, append = T)
      } else if (class(theObject)[1] == "Missingvalue")
      {
        cat(c(theObject@title, "в", "строке", row_index, "столбце", col_index, paste0("(", col_name,
        ")"), "\n"), file = myfile_report@file, append = T)
      } else
      {
        cat(c(theObject@title, "в", "строке", row_index, "столбце", col_index, paste0("(", col_name,
        ")"), ", значения ячейки =", paste0("(", value, ")", "\n"), file = myfile_report@file, append = T)
      }
    }
  )

  Misprint <- setClass("Misprint",
    contains = "Error"
  )

  setMethod(f = "initialize",
    signature = "Misprint",
    definition = function(.Object)
    {
      .Object@title <- c("Исправление")
      .Object@col_index_legend <- 2
      .Object@style <- c("misprint")
      return(.Object)
    }
  )

  setGeneric(name = "Find",
    def = function(theObject, missing_values, unsolved_misprints, myfile_in, myfile_out,
    myfile_report, column_class)
    {
      standardGeneric("Find")
    }
  )

  setMethod(f = "Find",
    signature = c("Misprint", "Missingvalue", "UnsolvedMisprint", "FileIn", "FileOut", "FileReport",
    "Discrete"),
    definition = function(theObject, missing_values, unsolved_misprints, myfile_in, myfile_out,
    myfile_report, column_class)
    {
      c <- myfile_in@table[[column_class@column_index]]
      misprints_row_ind <- c()
      misprints_new_row_ind <- c()

      for (i in 1:length(c))
      {
        found <- FALSE
        if (is.na(c[i]) == TRUE)
        {
          misprints_row_ind <- i
          misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
          myfile_out@row_table_legend
          missing_values@indices <- append(missing_values@indices, values =
          paste(misprints_new_row_ind, column_class@column_index, sep = "."))
        }
      }
    }
  )

```

```

        PrintReport(missing_values, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], value_new = NULL)
        cat("Missing value coordinates are ", paste(i, column_class@column_index, sep = "."), "\n")
        next
    }
    for(j in 1:length(column_class@value))
    {
        if (found)
            break
        for(k in 1:length(column_class@value[[j]]))
        {
            if (toupper(c[i]) == toupper(column_class@value[[j]][k]))
            {
                found <- TRUE
                break
            }
        }
    }
    if (found)
        next
    if (length(column_class@key) != 0)
    {
        for(a in 1:length(column_class@key))
        {
            if (found)
                break
            for(b in 1:length(column_class@key[[a]]))
            {
                if (toupper(c[i]) == toupper(column_class@key[[a]][b]))
                {
                    found <- TRUE
                    misprints_row_ind <- i
                    misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
                    theObject@indices <- append(theObject@indices, values = paste(misprints_new_row_ind,
column_class@column_index, sep = "."))
                    myfile_out@table[[column_class@column_index]][i] <- unlist(column_class@value[[a]],
use.names = FALSE)
                    PrintReport(theObject, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], myfile_out@table[[column_class@column_index]][misprints_row_ind])
                    cat("Misprints coordinates are ", paste(i, column_class@column_index, sep = "."),
"\n")
                    break
                }
            }
        }
        if ((!found) & (a == length(column_class@key)))
        {
            misprints_row_ind <- i
            misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
            unsolved_misprints@indices <- append(unsolved_misprints@indices, values =
paste(misprints_new_row_ind, column_class@column_index, sep = "."))
            PrintReport(unsolved_misprints, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], value_new = NULL)
            cat("Unsolved misprint coordinates are", paste(i, column_class@column_index, sep =
"."), "\n")
            found <- TRUE
        }
    }
    if (!found)
    {
        misprints_row_ind <- i
        misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
        unsolved_misprints@indices <- append(unsolved_misprints@indices, values =
paste(misprints_new_row_ind, column_class@column_index, sep = "."))
        PrintReport(unsolved_misprints, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], value_new = NULL)
        cat("Unsolved misprint coordinates are", paste(i, column_class@column_index, sep = "."),
"\n")
        found <- TRUE
    }
}
output_list <- list("misprint" = theObject, "missingvalues" = missing_values ,
"unsolvedMisprint" = unsolved_misprints, "file" = myfile_out)
return(output_list)
}
)

setMethod(f = "Find",
signature = c("Misprint", "Missingvalue", "UnsolvedMisprint", "FileIn", "FileOut", "FileReport",
"Continuous"),
definition = function(theObject, missing_values, unsolved_misprints, myfile_in, myfile_out,
myfile_report, column_class)

```

```

{
  misprints_row_ind <- c()
  misprints_new_row_ind <- c()
  unsolved_number <- 0
  c <- myfile_in@table[[column_class@column_index]]
  pattern <- "^((\\d+)[,\\.]|([[:space:]])?(\\d+)?)"
  for (i in 1:length(c))
  {
    if (is.na(c[i]) == TRUE)
    {
      misprints_row_ind <- i
      misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
      missing_values@indices <- append(missing_values@indices, values =
paste(misprints_new_row_ind, column_class@column_index, sep = "."))
      PrintReport(missing_values, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], value_new = NULL)
      cat("Missing value coordinates are ", paste(i, column_class@column_index, sep = "."), "\n")
      next
    }

    if (grepl(pattern, c[[i]]) == FALSE)
    {
      misprints_row_ind <- i
      misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
      unsolved_misprints@indices <- append(unsolved_misprints@indices, values =
paste(misprints_new_row_ind, column_class@column_index, sep = "."))
      PrintReport(unsolved_misprints, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], value_new = NULL)
      cat("Unsolved misprint coordinates are ", paste(i, column_class@column_index, sep = "."),
"\n")
      unsolved_number <- unsolved_number + 1
      next
    } else if (grepl("[[:space:]]", c[[i]]) == TRUE)
    {
      misprints_row_ind <- i
      misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
      theObject@indices <- append(theObject@indices, values = paste(misprints_new_row_ind,
column_class@column_index, sep = "."))
      myfile_out@table[[column_class@column_index]][i] <- gsub("[[:space:]]", "",
myfile_in@table[[column_class@column_index]][i])
      PrintReport(theObject, myfile_report, misprints_new_row_ind, column_class@column_index,
colnames(myfile_in@table[column_class@column_index]), c[misprints_row_ind],
myfile_out@table[[column_class@column_index]][misprints_row_ind])
      cat("Misprints coordinates are ", paste(i, column_class@column_index, sep = "."), "\n")
    } else if (is.character(c[[i]]) && (grepl("[.]", c[[i]]) == TRUE))
    {
      misprints_row_ind <- i
      misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
      theObject@indices <- append(theObject@indices, values = paste(misprints_new_row_ind,
column_class@column_index, sep = "."))
      myfile_out@table[[column_class@column_index]][i] <- gsub("[.]", ",",
myfile_in@table[[column_class@column_index]][i])
      PrintReport(theObject, myfile_report, misprints_new_row_ind, column_class@column_index,
colnames(myfile_in@table[column_class@column_index]), c[misprints_row_ind],
myfile_out@table[[column_class@column_index]][misprints_row_ind])
      cat("Misprints coordinates are ", paste(i, column_class@column_index, sep = "."), "\n")
    }
  }
  as.numeric(file@table[[column_class@column_index]][i])
}
output_list <- list("misprint" = theObject, "missingvalues" = missing_values,
"unsolvedMisprint" = unsolved_misprints, "file" = myfile_out, "unsolved_number" = unsolved_number)
return(output_list)
}

)

setMethod(f = "Find",
signature = c("Misprint", "MissingValue", "UnsolvedMisprint", "FileIn", "FileOut", "FileReport",
"Dates"),
definition = function(theObject, missing_values, unsolved_misprints, myfile_in, myfile_out,
myfile_report, column_class)
{
  misprints_row_ind <- c()
  misprints_new_row_ind <- c()
  c <- myfile_in@table[[column_class@column_index]]
  pattern <- "^((\\d){2}([,\\.]|[-/])(\\d){2}([,\\.]|[-/])(\\d){2}|(\\d){4})$"

  for (i in 1:length(c))
  {
    if (is.na(c[i]) == TRUE)
    {
      misprints_row_ind <- i

```

```

        misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
        missing_values@indices <- append(missing_values@indices, values =
paste(misprints_new_row_ind, column_class@column_index, sep = "."))
        PrintReport(missing_values, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], value_new = NULL)
        cat("Missing value coordinates are ", paste(i, column_class@column_index, sep = "."), "\n")
    }
    next

    if (grepl(pattern, c[[i]]) == FALSE)
    {
        misprints_row_ind <- i
        misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
        unsolved_misprints@indices <- append(unsolved_misprints@indices, values =
paste(misprints_new_row_ind, column_class@column_index, sep = "."))
        PrintReport(unsolved_misprints, myfile_report, misprints_new_row_ind,
column_class@column_index, colnames(myfile_in@table[column_class@column_index]),
c[misprints_row_ind], value_new = NULL)
        cat("Unsolved misprint coordinates are ", paste(i, column_class@column_index, sep = "."),
"\n")
    }
    next
} else if (grepl("[,]|[-/]", c[[i]]) == TRUE)
{
    misprints_row_ind <- i
    misprints_new_row_ind <- misprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
    theObject@indices <- append(theObject@indices, values = paste(misprints_new_row_ind,
column_class@column_index, sep = "."))
    myfile_out@table[[column_class@column_index]][i] <- gsub("[,]|[-/]", ".",
myfile_in@table[column_class@column_index]][i])
    PrintReport(theObject, myfile_report, misprints_new_row_ind, column_class@column_index,
colnames(myfile_in@table[column_class@column_index]), c[misprints_row_ind],
myfile_out@table[[column_class@column_index]][misprints_row_ind])
    cat("Misprints coordinates are ", paste(i, column_class@column_index, sep = "."), "\n")
}

}

output_list <- list("misprint" = theObject, "missingvalues" = missing_values ,
"unsolvedMisprint" = unsolved_misprints, "file" = myfile_out)
return(output_list)
}
)

DateMisprint <- setClass("DateMisprint",
    contains = "Error"
)

setMethod(f = "initialize",
    signature = "DateMisprint",
    definition = function(.Object)
    {
        .Object@title <- c("Неупорядоченные даты")
        .Object@col_index_legend <- 5
        .Object@style <- c("dateMisprint")
        return(.Object)
    }
)

setGeneric(name = "Find",
    def = function(theObject, date1, date2, myfile_out, myfile_report)
    {
        standardGeneric("Find")
    }
)

setMethod(f = "Find",
    signature = "DateMisprint",
    definition = function(theObject, date1, date2, myfile_out, myfile_report)
    {
        dateMisprints_row_ind <- c()
        dateMisprints_new_row_ind <- c()
        columns <- sort(c(date1@column_index, date2@column_index))
        d1 <- as.Date(myfile_out@table[[columns[1]]], format = "%d.%m.%Y")
        d2 <- as.Date(myfile_out@table[[columns[2]]], format = "%d.%m.%Y")
        dateMisprints_row_ind <- append(dateMisprints_row_ind, which(d1 > d2))
        dateMisprints_new_row_ind <- dateMisprints_row_ind + myfile_out@row_header +
myfile_out@row_table_legend
        theObject@indices <- append(theObject@indices, values = outer(dateMisprints_new_row_ind,
columns, paste, sep = "."))
        for(i in 1:length(columns)) lapply(dateMisprints_new_row_ind, function(j)
PrintReport(theObject, myfile_report, j, columns[i], colnames(myfile_out@table[columns[i]]),
myfile_out@table[[columns[i]]][j], value_new = NULL))
        cat("Date misprints coordinates are ", paste(dateMisprints_new_row_ind, columns, sep = "."),
"\n")
        return(theObject)
    }
)

```

```

)
}
)
outlier <- setClass("Outlier",
  contains = "Error"
)

setMethod(f = "initialize",
  signature = "Outlier",
  definition = function(.Object)
{
  .Object@title<- c("Выбороч")
  .Object@col_index_legend<- 4
  .Object@style <- c("outlier")
  return(.Object)
}
)

setGeneric(name = "Find",
  def = function(theObject, myfile_out, myfile_report, column_class, unsolved_number)
{
  standardGeneric("Find")
}
)

setMethod(f = "Find",
  signature = "Outlier",
  definition = function(theObject, myfile_out, myfile_report, column_class, unsolved_number)
{
  outliers_row_ind <- c()
  outliers_new_row_ind <- c()
  only_digits <- c()
  outliers <- c()
  c <- myfile_out@table[[column_class@column_index]]

  if (unsolved_number != 0)
  {
    cat("It is impossible to determine outliers, there are unsolved misprints in the column ",
    column_class@column_index, "\n")
    PrintReport(theObject, myfile_report, row_index = NULL, column_class@column_index,
    colnames(myfile_out@table[column_class@column_index]), value = NULL, value_new = NULL, not_outliers
    = T)
  } else
  {
    for (i in 1:length(c))
    {
      if (is.na(c[i]) == TRUE)
      {
        next
      }

      if (grepl("^(\\d)+([.](\\d)+)?$", c[[i]]) == TRUE)
      {
        only_digits <- append(only_digits, c[i])
        next
      }
    }
    only_digits <- gsub("[,]", ".", only_digits)
    outliers <- boxplot.stats(as.numeric(only_digits))$out

    if (!is.null(outliers))
    {
      outliers_row_ind <- which(gsub("[,]", ".", c) %in% outliers, arr.ind = T, useNames = F)

      outliers_new_row_ind <- outliers_row_ind + myfile_out@row_header +
      myfile_out@row_table_legend
      theObject@indices <- append(theObject@indices, values = outer(outliers_new_row_ind,
      column_class@column_index, paste, sep = "."))
      for(i in 1:length(outliers_new_row_ind))
      {
        PrintReport(theObject, myfile_report, outliers_new_row_ind[i], column_class@column_index,
        colnames(myfile_out@table[column_class@column_index]), c[outliers_row_ind[i]], value_new = NULL)
      }
      cat("Outlier coordinates are ", paste(outliers_new_row_ind, column_class@column_index, sep
      = "."), "\n")
      print(c[outliers_row_ind])
    }
  }
  return(theObject)
}
)

MissingValue <- setClass("MissingValue",
  contains = "Error"
)

setMethod(f = "initialize",
  signature = "MissingValue",
  definition = function(.Object)

```

```
{  
  .Object@title<- "пропущенное значение")  
  .Object@col_index_legend<- 1  
  .Object@style <- c("missing_value")  
  return(.Object)  
}  
)
```