ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ГОРОДА МОСКВЫ
Юго-Западный административный округ
Государственное бюджетное общеобразовательное учреждение города Москвы
«Школа № 1533 «ЛИТ»

ВЫПУСКНОЙ ПРОЕКТ

учащихся группы 10.5
Ермаковой Светланы и Мамаевой Арины

Разработка кроссплатформенного приложения для визуализации динамических данных

Исполнитель:	Ермакова Светлана Денисовна
Заказчик:	Белянов Александр Александрович
Руководитель:	Белянов Александр Александрович
Консультанты:	Гиглавый Александр Владимирович
	Завриев Николай Константинович
	Терёхина Анна Олеговна
	Морозова Ирина Вениаминовна

Москва
2019

Содержание

1. Введение

По статистике, 60% людей воспринимают быстрее именно визуальную информацию. Мозгу легче понять, что ему хотят сказать, если представить данные в виде графиков и схем, к тому же, визуальные образы держатся в памяти намного дольше. Кроме того, визуализация помогает наводить порядок, преобразовывая сырые данные в наглядные диаграммы и графики. Без визуализации данных мы бы не смогли быстро и легко разбираться в таком огромном количестве данных, которые накапливаются сейчас каждый день. Ключевой идеей визуализации является поиск ответов на вопросы к данным в понятной и доступной форме.

Добавлено примечание ([AA1]): Здесь и далее: выравнивание по ширине (Ctrl + J).

Визуализация используется для выполнения следующих задач:

- обучения и разъяснения;
- решения проблем и выяснения истины;
- исследований и анализа;
- составления отчетов и презентаций.

Информация в графическом виде привлекает большее внимания, легче воспринимается и помогает быстрее донести до аудитории ваше сообщение. С помощью наглядных графиков и «дашбордов»[1] можно сделать понятными даже сложные наборы данных. И если вы хотите, чтобы большая часть ваших партнеров, коллег или клиентов могли взаимодействовать с вашими данными, нужно превратить скучные таблицы в красивые диаграммы. Однако динамическая визуализация имеет еще больше преимуществ, так как позволяет наблюдать за изменениями в данных за выбранный конечный промежуток времени.

Хотелось бы упомянуть человека, который одним из первых задумался о том, что может дать динамическая визуализация данных. Не найдя подходящих инструментов среди существующих инструментов для визуализации, шведский врач, академик, специалист по статистике и всемирно известный лектор Ханс Рослинг сам занялся изучением этой области, он является одним из создателей приложения Gapminder, используемого для иллюстрации данных.

[1] Дашборд - – вкладка для совмещения графиков

2. Постановка задачи, актуальность, целевая аудитория, аналоги

2.1 Постановка задачи

Задача данного проекта - создание кросс-платформенного приложения, которое совмещает в себе визуализацию динамических данных, понятный интерфейс и разнообразные средства инфографики.

Если говорить более подробно о концепции приложения, то она должна включать в себя следующие составляющие:

1. Разработка интерфейса подключения источников данных
    1.1. Поддержка Excel-файлов
    1.2. Возможность фильтрации данных
2. Разработка интерфейса добавления диаграмм
    2.1. Выбор цвета графика
    2.2. Возможность изменения заголовков
    2.3. Поддержка различных типов диаграмм:
        2.3.1. Bar Chart
        2.3.2. Spline Chart
        2.3.3. Pie Chart
3. Разработка интерфейса составления «дашборда» - вкладка для совмещения графиков
    3.1. В рамках «дашборда» — разработка интерфейса проигрывания динамических данных
        3.1.1. Отображение года для текущего графика во время процесса динамической визуализации
        3.1.2. Управление скоростью воспроизведения
        3.1.3. Управление направлением воспроизведения
        3.1.4. Возможность постановки на паузу
4. Разработка интерфейса сохранения построенного графика
5. Технические ограничения разработки проекта предполагают обязательную поддержку основных платформ: Windows, Linux
6. Использование языка Python

Дополнительные функции будут реализованы в последующих версиях приложения.

2.2 Актуальность

Что касается актуальности, то на данный момент нет инструмента, совмещающего в себе визуализацию динамических данных, понятный интерфейс, разнообразие средств

инфографики, который может принимать Excel файлы с любыми типами данных (например данные типа дата, большие числа, деньги, проценты, строки).

2.3 Целевая аудитория

Целевой аудиторией проектной работы являются любые учащиеся или научные сотрудники, которые нуждаются в анализе данных для своих работ и проектов.

Поскольку визуализация активно используется в бизнесе, разработанное приложение будет полезно и представителям бизнеса.

Добавлено примечание ([AA2]): Посмотрите, кого видят своими клиентами те же Tableau

2.3 Аналоги

Так как сегодня визуализация данных – это актуальная тема, то у нашего приложения существуют аналоги, но не один из них не совмещает в себе динамическую визуализацию и наличие «дашборда».

Программный аналог	Функционал			
	Визуализация динамических данных	**Широкий выбор инструментов**	**Удобный и понятный интерфейс**	**Наличие вкладки для совмещения графиков (дашборда)**
Tableau	-	+	+	+
Power BI	-	-+	-	+
Gapminder	+	-	+	-
Qlik Sense	-	-+	-	+
Excel	-	-	+	-
Google Data Studio	-	+-	+	+
Собственная разработка	+	-+	+	+

1) Tableau

Достоинства: широко применяется, поддерживает несколько платформ, над отчётом могут работать сразу несколько пользователей, а результатом можно поделиться по ссылке или почте. Пример работы программы представлен на (Рис. 2.3.1).

Недостатки: приложение дорогостоящее и данные необходимо предварительно структурировать.

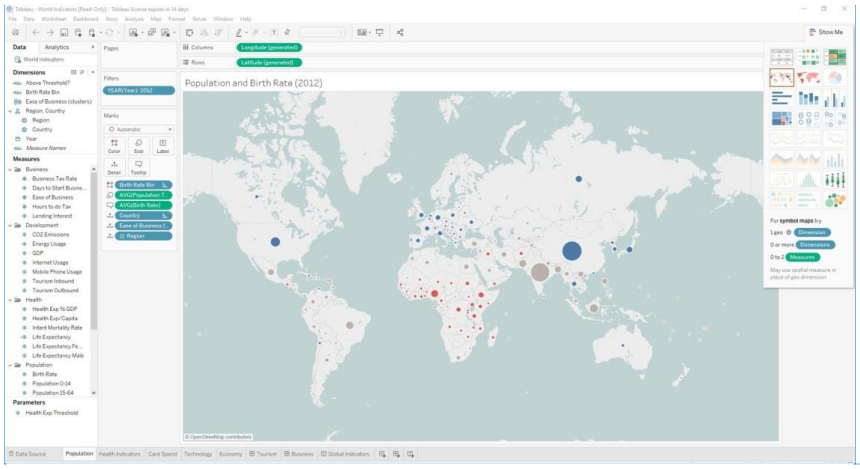


Рис. 2.3.1 Пример работы программы «Tableau».

Добавлено примечание ([АА3]): Подписи ко всем рисункам. Рисунки по центру.

2) Power BI

Преимущество: можно подключить практически любой офлайн и онлайн источник данных, а также у этого приложения полная интеграция с продуктами Microsoft, поддерживает несколько платформ (web, mobile, desktop).

Недостаток: у пользователей, которые не знакомы с Excel, могут возникнуть трудности при сложных операциях.

Пример работы программы представлен на (Рис. 2.3.2).

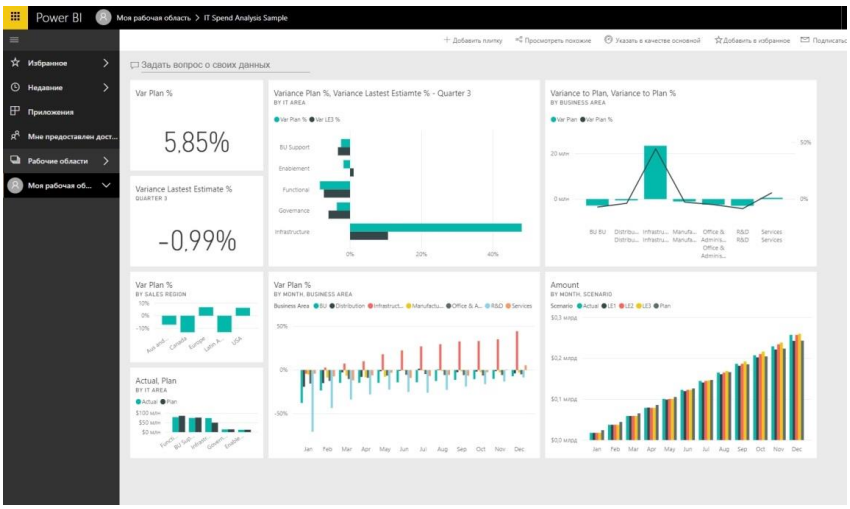


Рис. 2.3.2 Пример работы программы «Power BI».

3) Gapminder

Преимущество этой программы над всеми остальными аналогами в том, что в ней есть возможность наблюдения за изменением величин в динамике.

Недостатки: необходимость предварительного структурирования данных, отсутствие дашборда, выбор графиков ограничивается пятью типами.

Пример работы программы представлен на (Рис. 2.3.3).

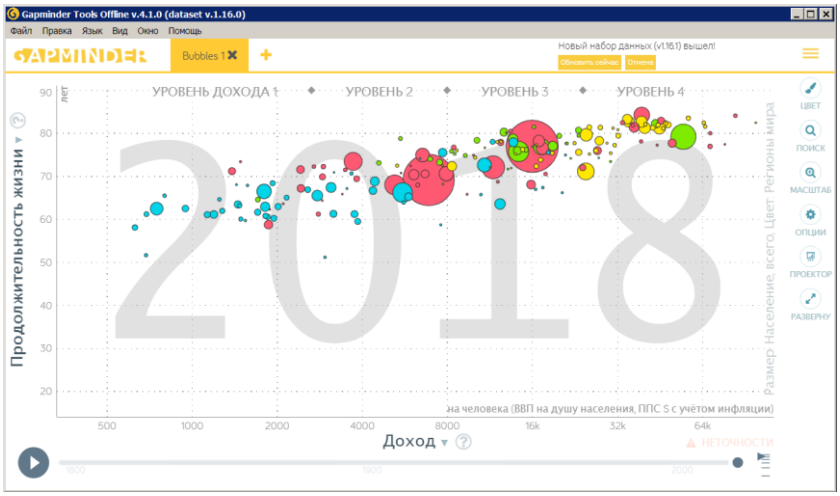


Рис. 2.3.3 Пример работы программы «Gapminder».

4) Qlik Sense

Преимущества: в этой программе лёгкая фильтрация данных для любой визуализации, а также быстрая скорость загрузки и обработки данных, создания графиков и таблиц.

Недостатки: программа дорогая и сложна в использовании.

Пример работы программы представлен на (Рис. 2.3.4).

Рис. 2.3.4 Пример работы программы «Qlik Sense».

5) Excel

Преимущества: входит в состав Microsoft Office и установлено практически на каждом компьютере.

Недостатки: эта программа неудобна для визуализации, так как прежде всего специализируется на анализе данных, поэтому создаёт графики по отдельности и не позволяет создать группировку графиков.

Пример работы программы представлен на (Рис. 2.3.5).

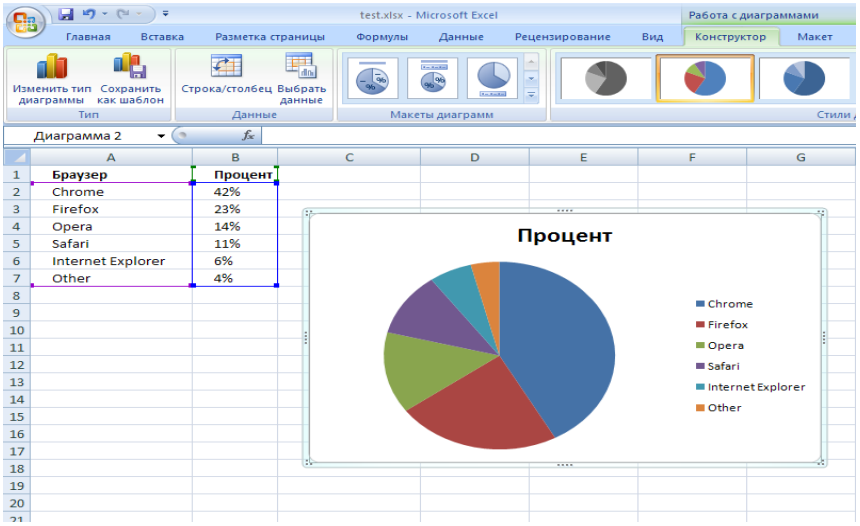


Рис. 2.3.5 Пример работы программы «Excel».

6) Google Data Studio

Преимущества: это бесплатный, просто и удобный инструмент, имеется галерея с готовыми дашбордами, сразу предлагает выбрать документы с текущего Google аккаунта и есть возможность группового одновременного редактирования.

Недостатки: намного более ограничен по функционалу, чем тот же Power BI

Пример работы программы представлен на (Рис. 2.3.6).

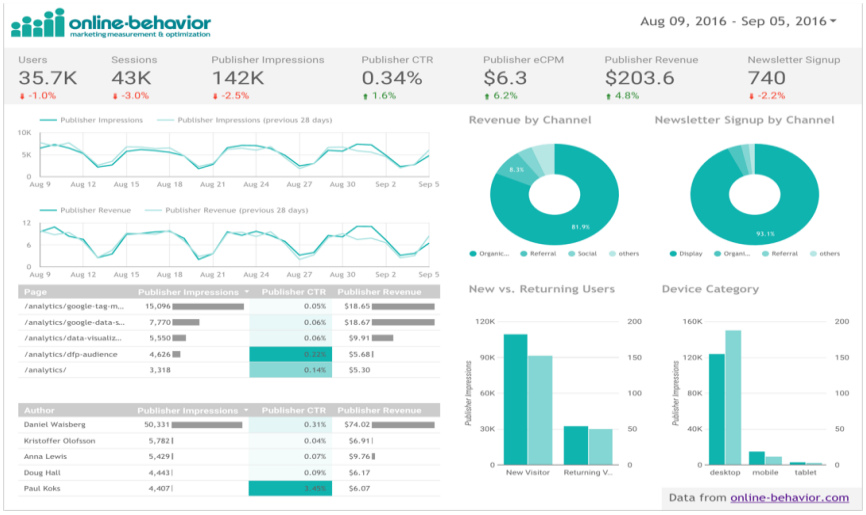


Рис. 2.3.6 Пример работы программы «Google Data Studio».

3. Решение

3.1. Теоретические выкладки

а) В нашем проекте для обозначения вкладки для совмещения графиков мы используем название – «дашборд». В переводе с английского «dashboard» - панель приборов. В бизнесе – это информационное табло, на котором отражаются важнейшие показатели, полученные в ходе обобщения данных.

б) В ходе реализации проекта мы решили подробно изучить 7 типов графиков:

1) Line Chart

Линейный график - самый простой из возможных и один из самых информативных графиков. Состоит из серии точек, соединенных горизонтальной линией. (Рис. 3.1)

Рис. 3.1

2) Bar Chart

Столбчатый график представлен прямоугольными зонами (столбцами), высоты или длины которых пропорциональны величинам, которые они отображают. Столбчатая диаграмма отображает сравнение нескольких дискретных категорий. Одна её ось показывает сравниваемые категории, другая — измеримую величину (Рис. 3.2).

Рис. 3.2

3) Spline Chart

Это особый тип линейных графиков, где отрезок между точкой сглаживается (Рис.3.3)

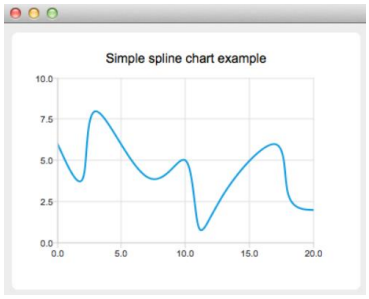


Рис. 3.3

4) Pie Chart

Круговая (секторная) диаграмма позволяет показать пропорциональное и процентное соотношение между категориями за счет деления круга на пропорциональные сегменты. Круг целиком представляет общую сумму всех данных, равную 100%. (Рис. 3.4)

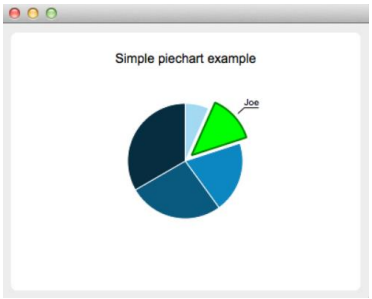


Рис. 3.4

5) Scatter Chart

Точечный график или график рассеяния изображает значения двух и более переменных в виде точек на декартовой плоскости, не соединяя их. Этот график помогает найти взаимосвязь между имеющимися показателями. (Рис. 3.5)

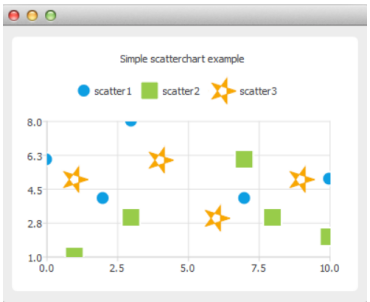


Рис. 3.5

6) Candlestick Chart

График «Японские свечи»— вид интервального графика, применяемый для отображения изменений биржевых котировок акций и цен на сырьё. Этот график состоит из тела и верхней/нижней тени (иногда говорят «фитиль"). Верхняя и нижняя граница тени отображает максимум и минимум цены за соответствующий период. Границы тела отображают цену открытия и закрытия. Свеча не содержит прямой информации о движении цен внутри соответствующего интервала времени. (Рис. 3.6)

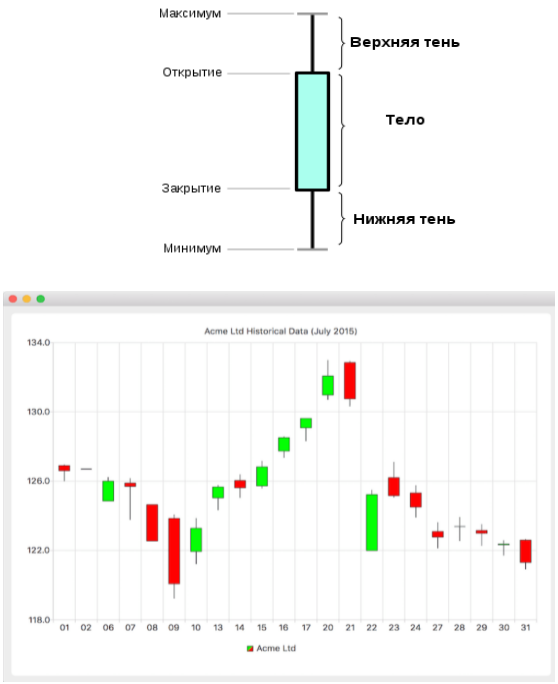


Рис. 3.6

7) Box and Whiskers Chart

График «Ящик с усами» используется в описательной статистике. Такой вид диаграммы в удобной форме показывает медиану, нижний и верхний квартили, минимальное и максимальное значение выборки и выбросы(отдельное значение). Расстояния между различными частями ящика позволяют определить степень разброса и асимметрии данных и выявить выбросы. Границами ящика служат первый и третий квартили(25% и 75%), линия в середине ящика — медиана (50%). Концы усов — края статистически значимой выборки (без выбросов). (Рис. 3.7)

Максимальное значение

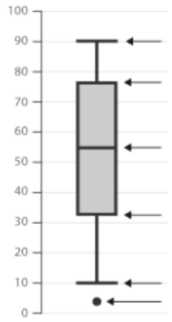
Верхний квартиль 75%

Медиана 50%

Нижний квартиль 25%

Минимальное значение

Выброс



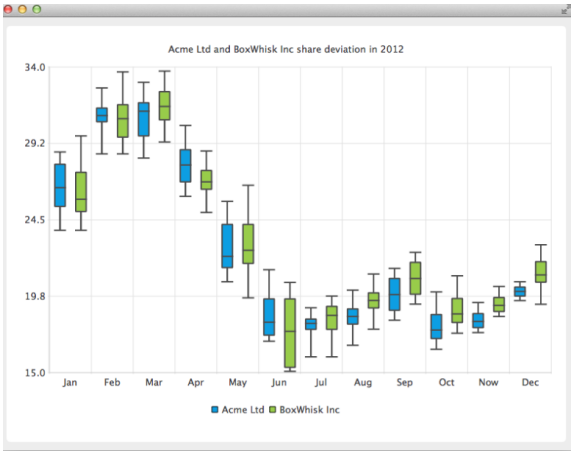


Рис. 3.7

3.2 Программная реализация

Одним из требований заказчика было то, что приложение должно быть кросс-платформенным, для реализации этой задачи был выбран язык программирования Python, согласно нашим исследованиям и информации заказчика именно на нем разработано большое количество бесплатных, богатых на возможности, библиотек. Кроме того, мы поняли, что это хороший обоснованный шанс освоить перспективный язык программирования. Среда разработки - Microsoft Visual Studio 2017. Несмотря на то, что изначально мы хотели использовать IDLE - базовую среду разработки программ для Python, спустя некоторое время мы решили вернуться к Visual Studio, так как хорошо освоили ее на уроках по специальности, а также в отличии от IDLE в нее интегрировано гораздо больше функций для реализации сложных проектов от этапа разработки до финального развертывания.

Для создания UML-диаграмм использована среда Microsoft Visio.

Анализ данных выполнялся с помощью Pandas. Для создания графического интерфейса нашего приложения и визуализации данных с помощью графиков мы используем PySide - адаптацию кроссплатформенного фрэймворка (комплекса множества библиотек) Qt под язык Python.

Для нашей программы нам понадобились:

- QtCore - основополагающий модуль, состоящий из классов, не связанных с графическим интерфейсом;
- QtGui - модуль базовых классов для программирования графического интерфейса;
- QtWidgets - модуль, дополняющий QtGui "строительным материалом" для графического интерфейса в виде виджетов;
- QtCharts - предоставляет набор простых в использовании компонентов диаграммы. Он использует Qt Graphics View Framework, поэтому графики могут быть легко интегрированы в современные пользовательские интерфейсы.

3.3 Ход работы

Из-за того, что существует большое количество аналогов, которые реализуют предполагаемый базовый функционал нашего приложения, нам пришлось довольно долго изучать детали его реализации в каждом из них. (на анализ и сравнение между собой аналогов ушло около двух с половиной месяцев).

Так как заказчик рекомендовал нам использовать Python, первое чем мы занялись после того, как смогли представить объем работы над программой, это изучение нового для нас языка (февраль-конец марта).

Параллельно с этим мы старались разобраться с функционалом выбранных библиотек (PySide2 и Pandas). На момент начала кодирования (начало апреля) мы еще не полностью овладели информацией о возможностях PySide2 и Pandas, поэтому основное время занимал поиск информации о том, как можно реализовать желаемое с их помощью.

Над первой тестовой версией программы мы работали вместе, она представляла собой доработанную версию кода, который мы нашли в описании видеоурока «Develop Your First Qt for Python Application». В ходе работы над ней мы старались разобраться как лучше распределить обязанности в работе над проектом и пришли к выводу, что гораздо полезнее будет, если мы сможем помогать и направлять друг друга по ходу работы, для чего нам надо вместе принимать все основополагающие решения по написанию кода:

1. Разработка общей идеи программы, которая бы реализовывала концепцию. приложения, описанную в Техническом Задании (пользователь может добавлять вкладки на «панель с вкладками», в зависимости от его выбора вкладка будет представлять из себя

Таблицу с данными, либо график выбранного типа с настройками, либо «дашборд - вкладку», на которой можно будет сравнить между собой графики, а так же наблюдать за их динамикой)

2. Разработка идеи реализации динамики на основе данных заказчика (данные-листы Excel-каждый лист представляет данные за определенный временной промежуток): организуем цикл для всех листов в файле, переходя на новый лист мы заново рисуем график по данным из этого листа, а предыдущий (нарисованный по данным предыдущего листа) удаляем, управляя направлением прохода по листам мы выбираем, в каком направлении пользователь увидит динамику(динамика вперед/назад), задавая время между двумя итерациями цикла мы регулируем скорость воспроизведения динамики).

Конкретизация моего круга задач описана ниже.

1) Более подробное изучение pandas. Цель - анализ данных (в том числе отбор столбцов с данными, в которых содержатся только числа, либо пустые значения/поиск пустых значений/поиск строковых и числовых значений/возможность исключения из данных столбцов с пустыми значениями, что важно для отбора из файла тех столбцов, которые могут предлагаться пользователю при выборе данных для графика). Представление данных в виде таблиц.

2) Реализация базовых функций «дашборда» (На «дашборде» находится список «вкладок с графиками», которые есть на данный момент в «панели с вкладками»; возможность увидеть конкретный график на «дашборде» по нажатию на него в списке «вкладок с графиками»; реализация динамики).

3) Реализация функции сохранения графика.

4) Усовершенствование классов графиков, написанных соавтором, чтобы пользователь мог самостоятельно выбирать столбцы с данными.

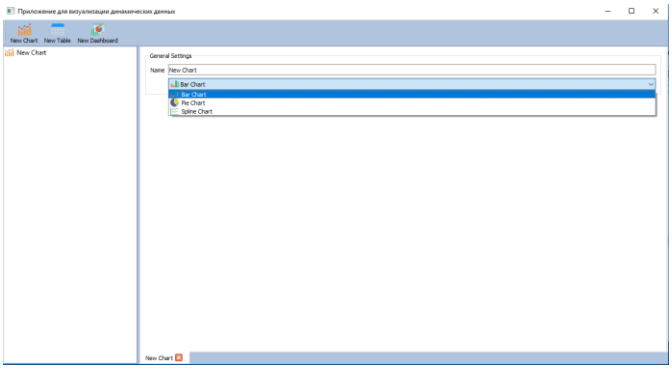
4. Результат

Результатом работы является программа, в которой реализованы пункты, описанные в постановке задачи.

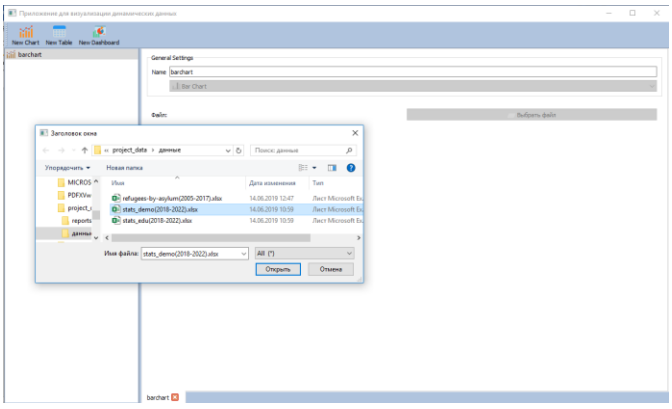
В связи с тем, что заказчиком был установлен приоритет по реализации динамики для ограниченного набора графиков, на данный момент помимо нашей основной программы (в которой реализована возможность наблюдать за динамикой изменения данных, представленных графиками barchart, splinechart, piechart) у нас есть пока находящаяся в разработке версия программы, в которой реализована возможность наблюдать за статической визуализацией данных с помощью графиков barchart, splinechart, piechart, candlestickchart, scatterchart, linechart.

Примеры скриншотов приложения:

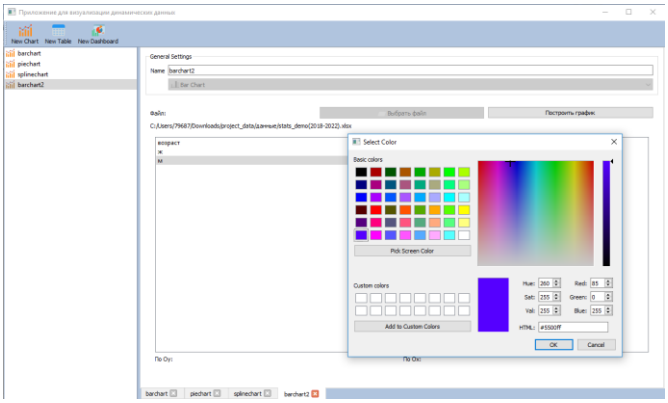
1. Выбор типа графика



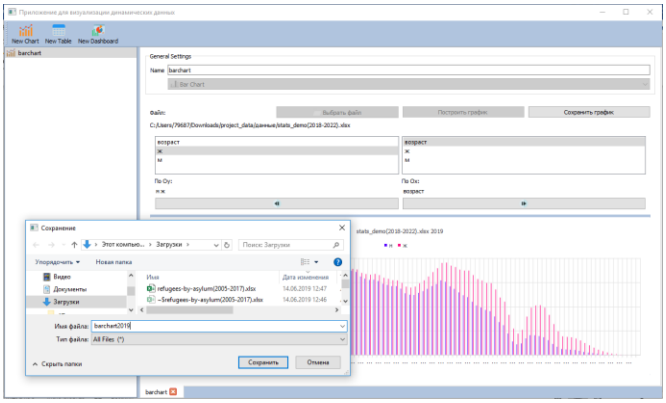
2. Выбор файла



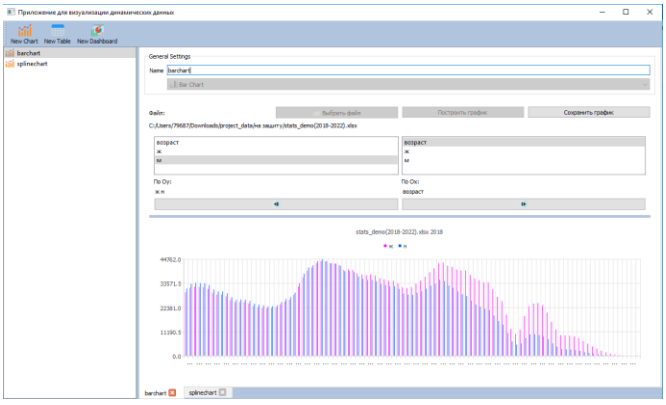
3. Выбор цвета для графика



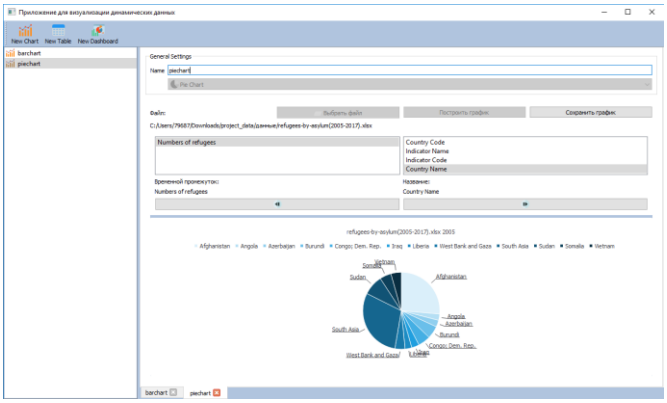
4. Сохранение графика



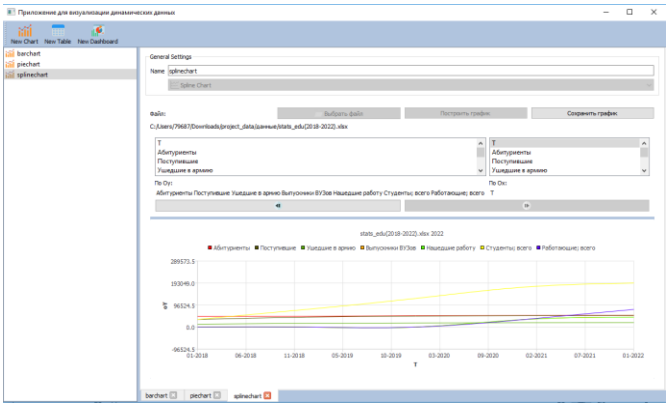
5. График типа barchart и его настройки



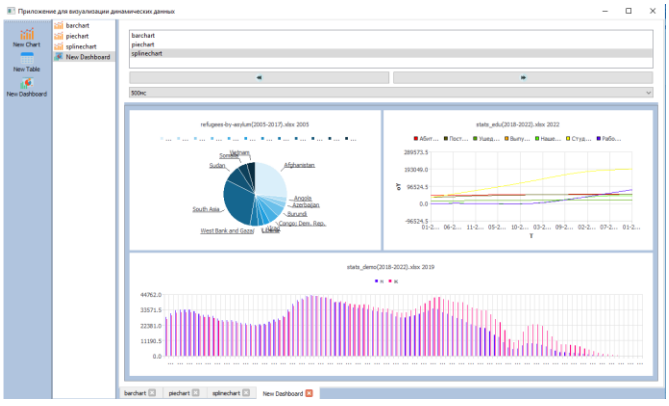
6. График типа piechart и его настройки



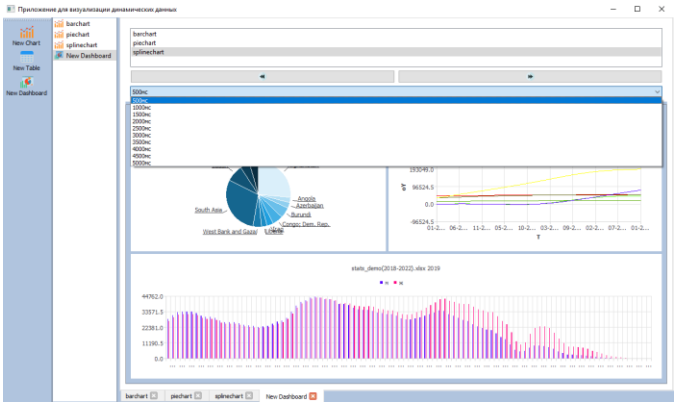
7. График типа splinechart и его настройки



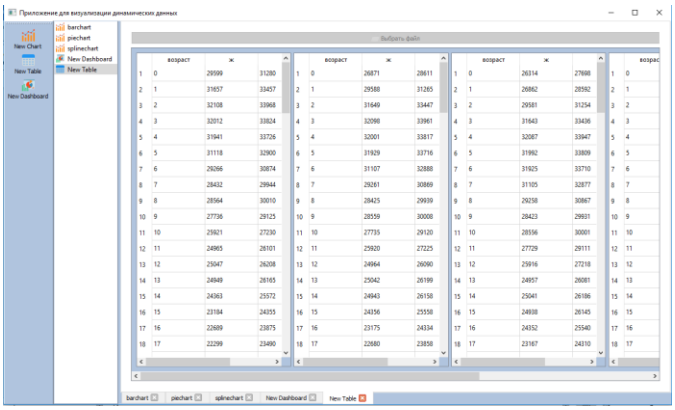
8. “Дашборд” с тремя графиками



9. Выбор времени



10. Таблица



Направления дальнейшей разработки

В последующих версиях проекта мы планируем:

1. Расширить выбор графиков и добавить кастомные диаграммы (диаграммы которые не входят в стандартные библиотеки Qt):

1.1. Area chart;

1.2. Horizontal bar chart;

1.3. Bubble chart;

1.4. Radar chart;

1.5. Polar chart;

1.6. Histogram;

1.7. Population Pyramid chart;

1.8. Maps.

2. Улучшить интерфейс и добавить темы - изменение цветов для визуальных элементов на «дашборде».

5. Выводы (заключение)

Работая над данным проектом, мы расширили свои знания по визуализации данных, познакомились с программами по анализу и визуализации данных, узнали про разные типы диаграмм и изучили новый для нас язык программирования - Python, который, возможно, пригодится нам в будущем. Нами получен огромный опыт работы с заказчиком и в команде.

В ходе реализации проекта мы на собственном опыте убедились насколько востребованным является наш продукт, получили уникальный опыт взаимодействия с потенциальными клиентами, выступив на семинаре Лаборатории социального моделирования ЦЭМИ РАН, где в ходе вопросов и ответов мы получили более полное представление о приоритетах дальнейшей разработки, о том как повысить востребованность нашего продукта, а так же представление об организации взаимодействия с клиентом в рамках работы над общим проектом.

Примером такого взаимодействия стал наш совместный проект с ЦЭМИ РАН по визуализации работы агентно-ориентированных моделей. Эти визуализации были представлены в рамках доклада научного руководителя ЦЭМИ РАН академика В.Л. Макарова на Президиуме Российской Академии Наук.

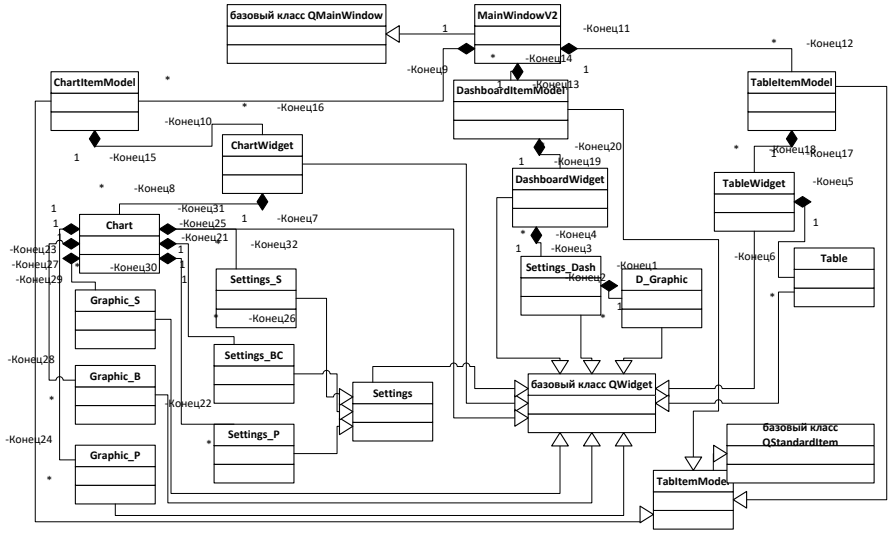
Разумеется, существует большой перечень функций, требующих реализации в дальнейшей работе, к ним можно отнести добавление новых типов графиков и улучшение интерфейса. Выполнение указанных пунктов будет являться важной задачей в ходе продолжения разработок в данной области.

6. Список литературы

1. Прохоренок, Н. А. Python 3 и PyQt 5. Разработка приложений. — 2-е изд., перераб. и доп. / Н. А. Прохоренок, В. А. Дронов. — СПб.: БХВ-Петербург, 2019. — 832с. - ISBN 978-5-9775-3978-4
2. Маккинли. Уэс Python и анализ данных / Уэс Маккинли ; пер. А. А. Слинкин. - Москва : ДМК Пресс, 2015. - 481 с. - ISBN 978-5-9706031-5-4
3. Widget Examples [Электронный ресурс]: Режим доступа: https://doc.qt.io/archives/qt-4.8/examples-widgets.html (дата обращения – 17.03.2019)
4. QDataTime Class [Электронный ресурс]: Режим доступа: https://doc.qt.io/qt-5/qdatetime.html#details (дата обращения – 22.03.2019)
5. Модуль QtCore [Электронный ресурс]: Режим доступа: http://doc.crossplatform.ru/qt/4.6.x/qtcore.html (дата обращения – 20.04.2019)
6. Qt Charts Examples [Электронный ресурс]: Режим доступа: https://doc.qt.io/qt-5/qtcharts-examples.html (дата обращения – 30.05.2019)
7. Перевод документации Python 3.x [Электронный ресурс]: Режим доступа: https://pythoner.name/documentation (дата обращения – 15.04.2019)
8. Python 3.7.3 documentation [Электронный ресурс]: Режим доступа: https://docs.python.org/3/ (дата обращения – 27.03.2019)
9. Введение в pandas: анализ данных на Python [Электронный ресурс]: Режим доступа: https://khashtamov.com/ru/pandas-introduction/ (дата обращения – 13.03.2019)
10. Develop Your First Qt for Python Application [Электронный ресурс]: Режим доступа: https://www.youtube.com/watch?v=HDBjmSiOBxY&feature=youtu.be (дата обращения – 09.01.2019)
11. Язык программирования PYTHON [Электронный ресурс]: Режим доступа: https://www.youtube.com/playlist?list=PLQAt0m1f9OHvv2wxPGSCWjgy1qER_FvB6 (дата обращения – 09.01.2019)

8. Приложения

a. Диаграмма классов



b. Диаграмма прецедентов


Актер1
-Конец1
-Конец2
-Конец3
-Конец4
-Конец5
-Конец6
Создать график
Создать дашборд
Создать таблицу
«uses»
«extends»
добавление вкладки графика со списком типов графиков
Создать splinechart
Создать barchart
Создать piechart
выбор файла с данными
Появление настроек графика
заполнение настроек графика и нажатие на кнопку добавить график
создание графика
добавление вкладки дашборда и на ней списка графиков кт есть на текущий момент
нажатие на график из списка
появление графика на дашборде и настроек дашборда
добавление вкладки таблицы
создание таблицы

```
from PySide2 import QtWidgets
from PySide2.QtWidgets import QMainWindow, QApplication, QWidget
from PySide2.QtWidgets import QAction
from PySide2.QtWidgets import QDockWidget, QListWidget, QTabWidget, QLabel, QLineEdit
from PySide2.QtWidgets import QListView, QGroupBox, QComboBox, QStackedWidget,
QSpacerItem
from PySide2.QtWidgets import QPushButton
from PySide2.QtWidgets import QFileDialog
from PySide2.QtWidgets import QGridLayout, QVBoxLayout
from PySide2.QtWidgets import QHeaderView,QHBoxLayout,  QSizePolicy, QTableView,
QSplitter,QScrollArea

from PySide2 import QtCore
from PySide2.QtCore import QStringListModel
from PySide2.QtCore import (QAbstractTableModel, QDateTime, QModelIndex, QTimeZone,
QPointF, Slot,QPoint)
from PySide2.QtCore import Qt

from PySide2 import QtGui
from PySide2.QtGui import QColor, QPainter,QPen, QLinearGradient, QGradient
from PySide2.QtGui import QStandardItemModel, QStandardItem
from PySide2.QtGui import QImageReader, QIcon

import PySide2
from PySide2.QtCharts import QtCharts

import sys
import argparse
import pandas as pd
import copy
import statistics
import sip
import time
```

```
import numpy as np

import pyautogui
import imutils
import cv2

class ChartWidget(QWidget):
    def __init__(self, model, tabWidget, parent=None):
        super(ChartWidget, self).__init__(parent)
        self.tabs_model=parent.model
        self.model = model
        self.tabWidget = tabWidget
        layout = QVBoxLayout(self)
        chartGroupBox = QGroupBox("General Settings")

        grid = QGridLayout(self)
        grid.addWidget(QLabel("Name"), 0, 0)
        self.nameLineEdit = QLineEdit()
        self.nameLineEdit.setText(model.text())
        self.nameLineEdit.textChanged.connect(self.updateText)

        grid.addWidget(self.nameLineEdit, 0, 1)
        self.typeComboBox = QComboBox()
        self.typeComboBox.activated.connect(self.changeType)

        grid.addWidget(self.typeComboBox, 1, 1)

        chartGroupBox.setLayout(grid)

        layout.addWidget(chartGroupBox)

        self.chartWidget = QWidget()
        layout.addWidget(self.chartWidget)

        self.stackedWidget = QStackedWidget(self)
```

```
    layout.addWidget(self.stackedWidget)

    self.setLayout(layout)

    chartTypesLst = [('Bar Chart', "icons/chart-types/bar-chart.svg", [Graphic_B,Settings_BC]),
        ('Pie Chart', "icons/chart-types/pie-chart.svg", [Graphic_P,Settings_P]),
        ('Spline Chart', "icons/chart-types/spline-chart.svg", [Graphic_S,Settings_S])]
    s = QStandardItemModel()
    for ct in chartTypesLst:
        item = QStandardItem(ct[0])
        item.setIcon(QIcon(ct[1]))
        item.setData(ct[2])
        s.appendRow(item)
    self.typeComboBox.setModel(s)
def changeType(self):
    self.typeComboBox.setEnabled(False)
    if self.stackedWidget.count() != 0:
        self.stackedWidget.removeWidget(self.stackedWidget.widget(0))

    row = self.typeComboBox.currentIndex()
    idx = self.typeComboBox.model().index(row, 0)
    model = self.typeComboBox.model().itemFromIndex(idx)
    t = model.data()
    if t == [Graphic_B,Settings_BC]:
        widget=Chart(t,0)
        self.stackedWidget.addWidget(widget)
    elif t == [Graphic_S,Settings_S]:
        widget = Chart(t,1)
        self.stackedWidget.addWidget(widget)
    elif t == [Graphic_P,Settings_P]:
        widget = Chart(t,2)
        self.stackedWidget.addWidget(widget)

def updateText(self):
    name = self.nameLineEdit.text()
```

```
        self.model.setText(name)
        ind = self.tabWidget.indexOf(self)
        self.tabWidget.setTabText(ind, name)

        for i in range(self.tabs_model.rowCount()):
            el=self.tabs_model.itemFromIndex(self.tabs_model.index(i, 0))
            if(el.model_name =='D'):
                el.tab.Settings_D.update_all_charts(self.tabs_model)

class Chart(QWidget):
    def __init__(self,arr,Type):
        QWidget.__init__(self)
        self.typeGr=arr[0]
        self.typeSt=arr[1]

        self.splitter = QSplitter(QtCore.Qt.Vertical)
        self.On_DB=False
        self.flag=False
        self.Type=Type

        self.chartView = None
        self.chartView_D =None

        self.button=QtWidgets.QPushButton("Построить график")
        self.button1=QtWidgets.QPushButton("Сохранить график")

        grid = QGridLayout(self)
        self.settingsGrid = QGridLayout(self)
        self.settingsGrid.addWidget(QLabel("Файл:"), 0, 0)
        self.datasetLabel = QLabel("нет данных")
        self.settingsGrid.addWidget(self.datasetLabel, 1, 0, 1, 2)
        self.dataSetButton = QPushButton("Выбрать файл", self)
        self.dataSetButton.setIcon(QIcon('icons/file.svg'))
        self.dataSetButton.clicked.connect(self.chooseDataset)
```

```
        self.settingsGrid.addWidget(self.dataSetButton, 0, 1)

        grid.addLayout(self.settingsGrid, 0, 0)
        grid.addWidget(self.splitter, 1, 0)
        self.setLayout(grid)

        self.button.clicked.connect(self.on_clicked_button)
        self.button1.clicked.connect(self.on_clicked_button1)
    def chooseDataset(self):
        self.dataSetButton.setEnabled(False)
        file = QtWidgets.QFileDialog.getOpenFileName(parent=window,
            caption="Заголовок окна", directory="c:\\",
            filter="All (*);;Exes (*.exe *.dll)",
            initialFilter="Exes (*.exe *.dll)")
        self.filename = file[0]
        self.datasetLabel.setText(self.filename)
        self.Settings_G = self.typeSt(self)
        self.splitter.addWidget(self.Settings_G)
    def on_clicked_button(self):
        if((self.Settings_G.criteria_x.text()!=' ') ==True &(self.Settings_G.criteria_y.text()!='
')==True):
            if(self.Type!=2):
                self.Settings_G.Max_Min()
            self.button.setEnabled(False)
            graphic = self.typeGr(self.Settings_G)
            self.chartView=graphic.chartView
            self.chartView.setMinimumSize(100, 300)

            self.Settings_G.main_layout.addLayout(self.Settings_G.layout_B)

            self.chartView_D=QtCharts.QChartView(QtCharts.QChart())
            self.splitter.addWidget(self.chartView)
            self.splitter.setCollapsible(self.splitter.count()-1,False)
            self.flag=True
```

```
    def on_clicked_button1(self):
        K1=self.splitter.widget(1).pos()
        size=self.splitter.widget(1).size()
        K2=QPoint(size.width(), size.height())

        size_b=self.Settings_G.Button_Back.size()
        KK1=self.mapToGlobal(K1)
        image = pyautogui.screenshot(region=(KK1.x(),KK1.y()+2*size_b.height(), K2.x(),K2.y()))

        image = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
        file = QtWidgets.QFileDialog.getSaveFileName(parent=window)
        path=file[0]
        if(path.endswith('.png')==False):
            path=path+'.png'
        cv2.imwrite(path, image)



class Settings(QWidget):
    def __init__(self, Chart):
        super(Settings, self).__init__()
        self.main_layout=QtWidgets.QVBoxLayout()
        Layout=QtWidgets.QHBoxLayout()
        self.layout1=QtWidgets.QVBoxLayout()
        self.layout2=QtWidgets.QVBoxLayout()
        self.layout_B=QtWidgets.QHBoxLayout()
        self.Chart=Chart
        self.Type=Chart.Type
        self.filename=Chart.filename
        self.normalname=self.normalN(self.filename)
        self.xls=None
        self.List_Number=0
        self.func_Data()

        self.Button_Forw=QtWidgets.QPushButton()
```

```
    self.Button_Forw.setIcon(QIcon('icons/forward.svg'))
    self.Button_Back=QtWidgets.QPushButton()
    self.Button_Back.setIcon(QIcon('icons/backwards.svg'))
    self.Button_Forw.clicked.connect(self.on_clicked_Button_Forw)
    self.Button_Back.clicked.connect(self.on_clicked_Button_Back)

    if(Chart.Type!=2):
        self.Color=[]

    self.numbers=None
    self.Columns=None

    Layout.addLayout(self.layout1)
    Layout.addLayout(self.layout2)
    self.layout_B.addWidget(self.Button_Back)
    self.layout_B.addWidget(self.Button_Forw)

    self.main_layout.addLayout(Layout)
    self.setLayout(self.main_layout)
def add_predstavlenie_y(self):

    self.Colums_model_y = QtCore.QStringListModel(self.numbers)
    self.criteria_y=QLabel(" ")
    self.Arr_Col_y=[]
    self.Colum_predstavl_y = QtWidgets.QListView()
    self.Colum_predstavl_y.setModel(self.Colums_model_y)

    self.layout1.addWidget(self.Colum_predstavl_y)
    if(self.Type==2):
        label=QLabel("Временной промежуток: ")
    elif(self.Type<2):
        label=QLabel("По Oy: ")
    self.layout1.addWidget(label)
    self.layout1.addWidget(self.criteria_y)
```

```
        self.Colum_predstavl_y.clicked.connect(self.Choice_y)
    def add_predstavlenie_x(self):
        self.Colums_model_x = QtCore.QStringListModel(self.Columns)

        self.criteria_x=QLabel(" ")
        self.Colum_predstavl_x = QtWidgets.QListView()

        self.Colum_predstavl_x.setModel(self.Colums_model_x)

        self.layout2.addWidget(self.Colum_predstavl_x)
        if(self.Type==2):
            label=QLabel("Название: ")
        elif(self.Type<2):
            label=QLabel("По Ox: ")
        self.layout2.addWidget(label)
        self.layout2.addWidget(self.criteria_x)

        self.Colum_predstavl_x.clicked.connect(self.Choice_x)
    def normalN(self, filename):
        index=filename.rfind('/')
        return filename[index+1:]
    def func_Data(self):
        if(self.filename[-1:]=='x'):
            self.xls = pd.ExcelFile(self.filename)
            self.Data = pd.read_excel(self.filename,
sheet_name=self.xls.sheet_names[self.List_Number])
    def on_clicked_Button_Forw(self):
        self.List_Number=self.List_Number+1
        if(self.List_Number>=len(self.xls.sheet_names)-1):
            self.Button_Forw.setEnabled(False)
        self.Data = pd.read_excel(self.filename, sheet_name=self.xls.sheet_names[self.List_Number])
        self.RePaint_G()
        if(self.Button_Back.isEnabled()==False):
            self.Button_Back.setEnabled(True)
    def on_clicked_Button_Back(self):
```

```
        self.List_Number=self.List_Number-1
        if(self.List_Number<=0):
            self.Button_Back.setEnabled(False)
        self.Data = pd.read_excel(self.filename, sheet_name=self.xls.sheet_names[self.List_Number])
        self.RePaint_G()
        if(self.Button_Forw.isEnabled()==False):
            self.Button_Forw.setEnabled(True)

class Graphic_B(QWidget):
    def __init__(self, Settings_G):
        QWidget.__init__(self)
        self.Settings_G=Settings_G
        self.Data=Settings_G.Data
        self.add_Graphic()
        Settings_G.Chart.settingsGrid.addWidget(Settings_G.Chart.button1, 0, 3)

    def add_Graphic(self):
        Yarr_c = self.Settings_G.Arr_Col_y
        Yarr_d=[]
        for i in range(len(Yarr_c)):
            Yarr_d.append(self.Data[Yarr_c[i]].tolist())
        X_c = self.Settings_G.Col_x
        X_d=self.Data[X_c].tolist()
        self.chart = self.createBarCharts(Yarr_d, Yarr_c, X_c, X_d)
        self.chart.setContentsMargins(-10, -10,-10,-10)

        self.chartView = QtCharts.QChartView(self.chart)
        self.chartView.setRenderHint(QPainter.Antialiasing)
    def createBarCharts(self, Yarr_d, Yarr_c, X_c, X_d):
        barSet=[]
        stroka=""
        for i in range(len(Yarr_c)):
            barSet.append(QtCharts.QBarSet(Yarr_c[i]))
            stroka=stroka+Yarr_c[i]+" "
            barSet[len(barSet)-1].append(Yarr_d[i])
```

```
            barSet[len(barSet)-1].setColor(self.Settings_G.Color[i])
        barSeries1 = QtCharts.QBarSeries()
        for u in range(len(barSet)):
            barSeries1.append(barSet[u])
        chart = QtCharts.QChart()
        chart.addSeries(barSeries1)
        chart.setTitle(self.Settings_G.normalname + "
"+self.Settings_G.xls.sheet_names[self.Settings_G.List_Number])

        categories = X_d
        barCategoryAxis = QtCharts.QBarCategoryAxis()
        barCategoryAxis.append(categories)
        chart.createDefaultAxes()
        chart.setAxisX(barCategoryAxis, barSeries1)

        chart.axes(Qt.Vertical)[0].setRange(self.Settings_G.MIN,self.Settings_G.MAX)
        chart.legend().setVisible(True)
        chart.legend().setAlignment(Qt.AlignTop)

        return chart
class Settings_BC(Settings):
    def __init__(self, Chart):
        super(Settings_BC, self).__init__(Chart)
        if(self.xls!= None):
            self.numbers=selection_Numbers_without_NULL(self.Data)
            self.add_predstavlenie_y()
            self.Columns=selection_Object_without_NULL(self.Data)
            self.add_predstavlenie_x()
            Chart.settingsGrid.addWidget(Chart.button, 0, 2)
    def RePaint_G(self):
        S=self.Chart.splitter.sizes()
        g=Graphic_B(self)
        self.Chart.chartView.setChart(g.chart)
        self.Chart.splitter.setSizes(S)
        if(self.Chart.On_DB==True):
```

```
            self.Repaint_D()
    def Repaint_D(self):
        g=Graphic_B(self)
        self.Chart.chartView_D.setChart(g.chart)
    def Choice_y(self):
        ff=self.Colum_predstavl_y.currentIndex()
        for i in range(len(self.numbers)):
            if(i==ff.row()):
                self.Color.append(PySide2.QtWidgets.QColorDialog.getColor())
                self.Arr_Col_y.append(self.numbers[i])
                str=self.criteria_y.text()
                self.criteria_y.setText(str+self.Arr_Col_y[len(self.Arr_Col_y)-1]+" ")
    def Choice_x(self):
        ff=self.Colum_predstavl_x.currentIndex()
        if(self.criteria_x.text()==" "):
            for i in range(len(self.Columns)):
                    if(i==ff.row()):
                        self.Col_x=self.Columns[i]
                        str=self.criteria_x.text()
                        self.criteria_x.setText(str+self.Col_x+" ")
                        break
    def Max_Min(self):
        Max=[]
        Min=[]
        if(self.Chart.filename[-1:]=='x'):
            xls=pd.ExcelFile(self.Chart.filename)
            for i in range(len(xls.sheet_names)):
                Data = pd.read_excel(self.Chart.filename, sheet_name=xls.sheet_names[i])
                for i in self.Arr_Col_y:
                    Max.append(max(Data[i].tolist()))
                    Min.append(min(Data[i].tolist()))
        self.MAX=max(Max)
        self.MIN=min(Min)

class Graphic_S(QWidget):
```

```
    def __init__(self, Settings_G):
        QWidget.__init__(self)
        self.Settings_G=Settings_G
        self.Data=Settings_G.Data
        self.add_Graphic()
        Settings_G.Chart.settingsGrid.addWidget(Settings_G.Chart.button1, 0, 3)
    def add_Graphic(self):
        Yarr_c = self.Settings_G.Arr_Col_y
        Yarr_d=[]
        for i in range(len(Yarr_c)):
            Yarr_d.append(self.Data[Yarr_c[i]].tolist())

        X_c = self.Settings_G.Col_x
        X_d=self.Data[X_c].tolist()

        data=self.getSplineChartData(Yarr_c, Yarr_d,X_c,X_d)
        self.chart = QtCharts.QChart()
        self.chart.setTitle(self.Settings_G.normalname + "
"+self.Settings_G.xls.sheet_names[self.Settings_G.List_Number])
        axisX = QtCharts.QDateTimeAxis();
        axisX.setFormat("MM-yyyy");
        axisX.setTitleText(X_c);
        self.chart.addAxis(axisX, Qt.AlignBottom);
        axisY = QtCharts.QValueAxis();
        axisY.setTitleText("oY");
        self.chart.addAxis(axisY, Qt.AlignLeft);
        Mi=self.Settings_G.MIN
        Ma=self.Settings_G.MAX

        self.chart.axes(Qt.Vertical)[0].setRange(Mi-(Ma-Mi)*0.5, Ma+(Ma-Mi)*0.5)
        valuesToDraw = [self.Data[i].tolist() for i in Yarr_c]
        namesToDraw = Yarr_c
        for i in range(len(namesToDraw)):
            values = valuesToDraw[i]
            name = namesToDraw[i]
```

```
            series = QtCharts.QSplineSeries()
            series.setName(name)
            series.setColor(self.Settings_G.Color[i])

            n = len(values)
            for j in range(n):
                time = data[0][j]
                series.append(float(time.toMSecsSinceEpoch()), values[j])
            self.chart.addSeries(series)
            series.attachAxis(axisX);
            series.attachAxis(axisY);
        self.chart.axes(Qt.Horizontal)[0].setRange(data[0][0],data[0][len(data[0])-1])
        self.chart.axes(Qt.Horizontal)[0].setTickCount(10)

        self.chart.setContentsMargins(-10, -10,-10,-10)

        self.chartView = QtCharts.QChartView(self.chart)
        self.chartView.setRenderHint(QPainter.Antialiasing)

    def getSplineChartData(self,Yarr_c, Yarr_d,X_c,X_d ):
        xls = pd.ExcelFile(self.Settings_G.filename)
        data = pd.read_excel(self.Settings_G.filename,
sheet_name=xls.sheet_names[len(xls.sheet_names)-1])
        timeStamps = data[X_c]
        timeStamps = [QDateTime.fromString(str(x),"yyyy")
                      for x in timeStamps]
        values = []
        names = []

        for x in self.Data.loc[:, Yarr_c[0]:]:
            values.append(list(self.Data[x]))
            names.append(x)
        return timeStamps, values, names
class Settings_S(Settings):
    def __init__(self, Chart):
```

```
    super(Settings_S, self).__init__(Chart)
    if(self.xls!=None):
        self.numbers=selection_Numbers_without_NULL(self.Data)
        self.add_predstavlenie_y()
        self.Columns=selection_Object_without_NULL(self.Data)
        self.add_predstavlenie_x()
        Chart.settingsGrid.addWidget(Chart.button, 0, 2)
def RePaint_G(self):
    S=self.Chart.splitter.sizes()
    g=Graphic_S(self)
    self.Chart.chartView.setChart(g.chart)
    self.Chart.splitter.setSizes(S)
    if(self.Chart.On_DB==True):
        self.Repaint_D()
def Repaint_D(self):
    g=Graphic_S(self)
    self.Chart.chartView_D.setChart(g.chart)
def Choice_y(self):
    ff=self.Colum_predstavl_y.currentIndex()
    for i in range(len(self.numbers)):
        if(i==ff.row()):
            self.Color.append(PySide2.QtWidgets.QColorDialog.getColor())
            self.Arr_Col_y.append(self.numbers[i])
            str=self.criteria_y.text()
            self.criteria_y.setText(str+self.Arr_Col_y[len(self.Arr_Col_y)-1]+" ")
def Choice_x(self):
    ff=self.Colum_predstavl_x.currentIndex()
    if(self.criteria_x.text()==" "):
        for i in range(len(self.Columns)):
                if(i==ff.row()):
                    self.Col_x=self.Columns[i]
                    str=self.criteria_x.text()
                    self.criteria_x.setText(str+self.Col_x+" ")
                    break
def Max_Min(self):
```

```
        Max=[]
        Min=[]
        if(self.Chart.filename[-1:]=='x'):
            xls=pd.ExcelFile(self.Chart.filename)
            for i in range(len(xls.sheet_names)):
                Data = pd.read_excel(self.Chart.filename, sheet_name=xls.sheet_names[i])
                for i in self.Arr_Col_y:
                    Max.append(max(Data[i].tolist()))
                    Min.append(min(Data[i].tolist()))
        self.MAX=max(Max)
        self.MIN=min(Min)

class Graphic_P(QWidget):
    def __init__(self, Settings_G):
        QWidget.__init__(self)
        self.Settings_G=Settings_G
        self.Data=Settings_G.Data
        self.add_Graphic()
        Settings_G.Chart.settingsGrid.addWidget(Settings_G.Chart.button1, 0, 3)
    def add_Graphic(self):
        Y_c = self.Settings_G.Col_y
        Y_d=self.Data[Y_c].tolist()
        X_c = self.Settings_G.Col_x
        X_d=self.Data[X_c].tolist()

        data=self.getPieChartData(X_c, Y_c)

        series = QtCharts.QPieSeries()
        sliceList = list(map(lambda nv: QtCharts.QPieSlice(nv[0], nv[1]),
                             zip(data[0], data[1])))
        series.append(sliceList)
        for s in series.slices():
            s.setLabelVisible()
        self.chart = QtCharts.QChart();
```

```
        self.chart.setTitle(self.Settings_G.normalname + "
"+self.Settings_G.xls.sheet_names[self.Settings_G.List_Number])

        self.chart.addSeries(series);
        self.chart.setContentsMargins(-10, -10,-10,-10)

        self.chartView = QtCharts.QChartView(self.chart)
        self.chartView.setRenderHint(QPainter.Antialiasing);
    def getPieChartData(self,X_c, Y_c):
        self.Data = self.Data[[X_c,Y_c]]
        s = self.Data[Y_c].sum()
        threshold = s / 100
        threshold *= 2
        self.Data = self.Data.loc[self.Data[Y_c] > threshold]

        names = list(self.Data[X_c])
        no = list(self.Data[Y_c])
        return names, no
class Settings_P(Settings):
    def __init__(self, Chart):
        super(Settings_P, self).__init__(Chart)
        if(self.xls!=None):
            self.numbers=selection_Numbers(self.Data)
            self.add_predstavlenie_y()
            self.Columns=selection_Object_without_NULL(self.Data)
            self.add_predstavlenie_x()
            Chart.settingsGrid.addWidget(Chart.button, 0, 2)
    def RePaint_G(self):
        S=self.Chart.splitter.sizes()
        g=Graphic_P(self)
        self.Chart.chartView.setChart(g.chart)
        self.Chart.splitter.setSizes(S)
        if(self.Chart.On_DB==True):
            self.Repaint_D()
    def Repaint_D(self):
```

```
        g=Graphic_P(self)
        self.Chart.chartView_D.setChart(g.chart)
    def Choice_x(self):
        ff=self.Colum_predstavl_x.currentIndex()
        if(self.criteria_x.text()==" "):
            for i in range(len(self.Columns)):
                if(i==ff.row()):
                    self.Col_x=self.Columns[i]
                    self.criteria_x.setText(self.Col_x)
                    break
    def Choice_y(self):
        ff=self.Colum_predstavl_y.currentIndex()
        if(self.criteria_y.text()==" "):
            for i in range(len(self.numbers)):
                if(i==ff.row()):
                    self.Col_y=self.numbers[i]
                    self.criteria_y.setText(self.Col_y)
                    break


def selection_Numbers_without_NULL(Data):
        arr=[]
        numerics1=['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
        ddd=Data.select_dtypes(include=numerics1)
        newdf=ddd.isnull().any()
        for i in range(len(newdf.values)):
            if(newdf.values[i]==False):
                arr.append(ddd.columns.values[i])
        return arr
def selection_Numbers(Data):
        arr=[]
        numerical1=['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
        numbers_null=Data.select_dtypes(include=numerical1)
        for i in range(len(numbers_null.columns.values)):
            column=numbers_null.columns.values[i]
            if(numbers_null[column].count()!=0):
```

```
            arr.append(column)
        return arr
def selection_Object_without_NULL(Data):
        arr=[]
        numerics1=['object','int16', 'int32', 'int64', 'float16', 'float32', 'float64']
        ddd=Data.select_dtypes(include=numerics1)
        newdf=Data.isnull().any()
        for i in range(len(newdf.values)):
           if(newdf.values[i]==False):
               arr.append(ddd.columns.values[i])
        return arr

class TableWidget(QWidget):
    def __init__(self, model, parent=None):
        super(TableWidget, self).__init__(parent)
        self.model = model

        layout = QVBoxLayout()
        table=Table()
        layout.addWidget(table)

        self.setLayout(layout)
class Table(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)

        self.main_layout = QVBoxLayout()
        self.splitter = QSplitter()
        self.splitter.setChildrenCollapsible(False)
        self.widg=QWidget()
        l= QVBoxLayout()
        l.addWidget(self.splitter)
        self.widg.setLayout(l)

        self.button=QtWidgets.QPushButton("Выбрать файл")
```

```
    self.button.setIcon(QIcon('icons/file.svg'))
    self.button.setCheckable(True)

    self.scrollArea=QScrollArea()

    self.scrollArea.setWidget(self.widg)
    self.scrollArea.setWidgetResizable(True)

    self.main_layout.addWidget(self.button)
    self.main_layout.addWidget(self.scrollArea)
    self.setLayout(self.main_layout)

    self.button.clicked.connect(self.on_clicked_button)
def on_clicked_button(self):
    file = QtWidgets.QFileDialog.getOpenFileName(parent=window,
        caption="Заголовок окна", directory="c:\\",
        filter="All (*);;Exes (*.exe *.dll)",
        initialFilter="Exes (*.exe *.dll)")
    filename = file[0]
    if(filename[-1:]=='x'):
        self.button.setEnabled(False)
        xls = pd.ExcelFile(filename)
        for i in range(len(xls.sheet_names)):
            Data = pd.read_excel(filename, sheet_name=xls.sheet_names[i])
            f=6
            self.build_table(Data)
            f=7
        self.widg.setMinimumSize(len(xls.sheet_names)*len(Data.columns.values)*100, 100)
    elif(filename==''):
        self.button.setChecked(False)
def build_table(self,Data):
    cc=Data.columns.values
    gg=Data[cc[1]].values
    tv2 = QtWidgets.QTableView()
    sti2 = QtGui.QStandardItemModel()
```

```
        mas_column=[]
        for r in range(len(cc)):
            for y in range(len(Data)):
                mas_column.append(QtGui.QStandardItem(str(Data[cc[r]].values[y])))
            sti2.appendColumn(mas_column)
            mas_column.clear()
        sti2.setHorizontalHeaderLabels(Data.columns)
        tv2.setModel(sti2)
        self.splitter.addWidget(tv2)

class DashboardWidget (QWidget):
    def __init__(self, model, parent=None):
        super(DashboardWidget, self).__init__(parent)
        main_layout = QtWidgets.QVBoxLayout()

        self.model = model
        self.tabs_model=parent.model

        self.splitter = QSplitter(QtCore.Qt.Vertical)

        self.Settings_D = Settings_Dash(self)
        self.D_Graphics= D_Graphic(self.Settings_D)
        self.scrollArea=QScrollArea()

        self.scrollArea.setMinimumSize(100, 400)
        self.scrollArea.setWidget(self.D_Graphics)
        self.scrollArea.setWidgetResizable(True)

        self.splitter.addWidget(self.Settings_D)
        self.splitter.addWidget(self.scrollArea)
        self.splitter.setCollapsible(1, False)

        main_layout.addWidget(self.splitter)

        self.setLayout(main_layout)
```

```
class Settings_Dash(QWidget):
    def __init__(self, dashboard):
        QWidget.__init__(self)
        self.main_layout = QtWidgets.QVBoxLayout()
        self.layout=QtWidgets.QHBoxLayout()

        self.D =dashboard
        self.tabs_model=dashboard.tabs_model
        self.all_charts =QListView()
        self.update_all_charts(self.tabs_model)

        self.Setting_SH=[]

        self.Time=500
        self.combobox=QComboBox()


        model = ['500мс',
'1000мс','1500мс','2000мс','2500мс','3000мс','3500мс','4000мс','4500мс','5000мс']
        s1 = QtCore.QStringListModel(model)
        self.combobox.setModel(s1)
        self.combobox.activated.connect(self.time_for_CB)

        self.button1=QPushButton()
        self.button1.setIcon(QIcon('icons/dynamic_forw.svg'))
        self.button1.setCheckable(True)

        self.button2=QPushButton()
        self.button2.setIcon(QIcon('icons/dynamic_back.svg'))
        self.button2.setCheckable(True)

        self.layout.addWidget(self.button2)
        self.layout.addWidget(self.button1)

        self.main_layout.addWidget(self.all_charts)
```

```
    self.setLayout(self.main_layout)

    self.button1.clicked.connect(self.on_clicked_button1)
    self.button2.clicked.connect(self.on_clicked_button2)
    self.all_charts.clicked.connect(self.cbo_on_clicked)

def update_all_charts(self, tabs):
    s = QStandardItemModel()
    for i in range(tabs.rowCount()):
        el=tabs.itemFromIndex(tabs.index(i, 0))
        if(el.model_name =='C'):
            item = QStandardItem(el.text())
            item.setData(el.tab)
            s.appendRow(item)
    self.all_charts.setModel(s)
def cbo_on_clicked(self):
    ff=self.all_charts.currentIndex()
    tt=self.all_charts.model().itemFromIndex(ff).data()
    if(tt.stackedWidget.widget(0).flag==True):
        if(tt.stackedWidget.widget(0).On_DB!=True):
            self.Setting_SH.append(tt.stackedWidget.widget(0))
            self.D.D_Graphics.add_Graphic_on_DB(tt.stackedWidget.widget(0))
            tt.stackedWidget.widget(0).On_DB=True
            if(self.main_layout.indexOf(self.button1)==-1):
                self.main_layout.addLayout(self.layout)
                self.main_layout.addWidget(self.combobox)
def on_clicked_button1(self):
    value=self.button1.isChecked()
    if(value==True):
        self.button1.setChecked(True)
        self.button1.setIcon(QIcon('icons/pause.svg'))

        self.timer_id = self.startTimer(self.Time)
        self.combobox.setEnabled(False)
```

```
        if(self.button2.isEnabled()==True):
            self.button2.setEnabled(False)
    elif(value==False):

        self.button1.setChecked(False)
        self.button1.setIcon(QIcon('icons/dynamic_forw.svg'))

        self.killTimer(self.timer_id)
        self.combobox.setEnabled(True)

        if(self.button2.isEnabled()==False):
            self.button2.setEnabled(True)

def on_clicked_button2(self):
    value=self.button2.isChecked()
    if(value==True):
        self.button2.setChecked(True)
        self.button2.setIcon(QIcon('icons/pause.svg'))
        self.timer_id = self.startTimer(self.Time)
        self.combobox.setEnabled(False)
        if(self.button1.isEnabled()==True):
            self.button1.setEnabled(False)
    elif(value==False):
        self.button2.setChecked(False)
        self.button2.setIcon(QIcon('icons/dynamic_back.svg'))

        self.killTimer(self.timer_id)
        self.combobox.setEnabled(True)

        if(self.button1.isEnabled()==False):
            self.button1.setEnabled(True)
def time_for_CB(self):
    text=self.combobox.currentText()
    index=text.find('м')
    self.Time=float(text[:index])
```

```
    def timerEvent(self, event):
        S1=self.D.D_Graphics.splitter1.sizes()
        S2=self.D.D_Graphics.splitter2.sizes()
        S3=self.D.D_Graphics.splitter3.sizes()
        arr=self.Setting_SH
        for u in arr:
            if(self.button1.isChecked()==True):
                if(u.Settings_G.List_Number>=len(u.Settings_G.xls.sheet_names)-1):
                    u.Settings_G.List_Number=-1
                    u.Settings_G.Button_Forw.setEnabled(True)
                u.Settings_G.Button_Forw.click()
            elif(self.button2.isChecked()==True):
                if(u.Settings_G.List_Number<=0):
                    u.Settings_G.List_Number=len(u.Settings_G.xls.sheet_names)
                    u.Settings_G.Button_Back.setEnabled(True)
                u.Settings_G.Button_Back.click()
            if(u.Type==0):
                g=Graphic_B(u.Settings_G)
            elif(u.Type==1):
                g=Graphic_S(u.Settings_G)
            elif(u.Type==2):
                g=Graphic_P(u.Settings_G)
            u.chartView_D.setChart(g.chart)
        self.D.D_Graphics.splitter1.setSizes(S1)
        self.D.D_Graphics.splitter2.setSizes(S2)
        self.D.D_Graphics.splitter3.setSizes(S3)
class D_Graphic(QWidget):
    def __init__(self, Settings_D):
        QWidget.__init__(self)
        main_layout = QHBoxLayout()
        self.Settings_D = Settings_D

        self.splitter3 = QSplitter(QtCore.Qt.Vertical)
        self.splitter1 = QSplitter()
        self.splitter2 = QSplitter()
```

```
        self.splitter3.addWidget(self.splitter1)
        self.splitter3.addWidget(self.splitter2)

        main_layout.addWidget(self.splitter3)
        self.setLayout(main_layout)
    def add_Graphic_on_DB(self,grapf):
        if(grapf.Type==0):
            g=Graphic_B(grapf.Settings_G)
        elif(grapf.Type==1):
            g=Graphic_S(grapf.Settings_G)
        elif(grapf.Type==2):
            g=Graphic_P(grapf.Settings_G)
        grapf.chartView_D.setChart(g.chart)
        w=grapf.chartView_D
        if(self.splitter1.count()%2==self.splitter2.count()%2):
            self.splitter1.addWidget(w)
        else:
            self.splitter2.addWidget(w)

class TabItemModel(QStandardItem):
    def __init__(self):
        super(TabItemModel, self).__init__()
        self.widget = None
        self.tab = None
        self.model_name=''

class ChartItemModel(TabItemModel):
    def __init__(self):
        super(ChartItemModel, self).__init__()
        self.setIcon(QIcon('icons/chart.svg'))
        self.setEditable(False)
        self.tab = None
        self.model_name = 'C'
class TableItemModel(TabItemModel):
```

```
    def __init__(self):
        super(TableItemModel, self).__init__()

        self.setIcon(QIcon('icons/table.svg'))
        self.setEditable(False)

        self.tab = None
        self.model_name = 'T'
class DashboardItemModel(TabItemModel):
    def __init__(self):
        super(DashboardItemModel, self).__init__()

        self.setIcon(QIcon('icons/dashboard.svg'))
        self.setEditable(False)

        self.tab = None
        self.model_name = 'D'

class MainWindowV2(QMainWindow):
    def __init__(self):
        super(MainWindowV2, self).__init__()
        p = self.palette()
        p.setBrush(self.backgroundRole(), QtGui.QBrush(QColor("#B0C4DE")) )
        self.setPalette(p)

        self.model = QStandardItemModel(self)

        self.tabWidget = QTabWidget(self)
        self.tabWidget.setTabPosition(QTabWidget.South)
        self.tabWidget.setTabsClosable(True)
        self.tabWidget.setMovable(True)

        self.tabWidget.tabCloseRequested.connect(self.closeTabFromTabWidget)
        self.setCentralWidget(self.tabWidget)
```

```
    logDockWidget = QDockWidget("Tabs", self)
    logDockWidget.setTitleBarWidget(QWidget())
    logDockWidget.setObjectName("LogDockWidget")
    logDockWidget.setAllowedAreas(Qt.LeftDockWidgetArea | Qt.RightDockWidgetArea)
    logDockWidget.setFeatures(QDockWidget.DockWidgetMovable)

    self.navListView = QListView()
    self.navListView.setModel(self.model)

    logDockWidget.setWidget(self.navListView)
    self.addDockWidget(Qt.LeftDockWidgetArea, logDockWidget)

    self.createActions()

    tabsToolbar = self.addToolBar("Tabs")
    tabsToolbar.setObjectName("tabsToolBar")
    tabsToolbar.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)

    for action in [self.newChartAction, self.newTableAction, self.newDashboardAction]:
        tabsToolbar.addAction(action)

    self.navListView.selectionModel().selectionChanged.connect(self.setActiveTab)
    self.newChart()
def newChart(self):
    item = ChartItemModel()
    item.setText('New Chart')

    self.model.appendRow(item)
    widget = ChartWidget(item, self.tabWidget, self)
    self.tabWidget.addTab(widget, "New Chart")
    item.tab = widget

    for i in range(self.model.rowCount()):
        el=self.model.itemFromIndex(self.model.index(i, 0))
        if(el.model_name =='D'):
```

```
            el.tab.Settings_D.update_all_charts(self.model)
        pass

    def newTable(self):
        item = TableItemModel()
        item.setText('New Table')
        self.model.appendRow(item)

        widget = TableWidget(item, self)
        self.tabWidget.addTab(widget, "New Table")

        item.tab = widget
        pass

    def newDashboard(self):
        flag=True
        for i in range(self.model.rowCount()):
            el=self.model.itemFromIndex(self.model.index(i, 0))
            if(el.model_name =='D'):
                flag=False
        if(flag):
            item = DashboardItemModel()
            item.setText('New Dashboard')
            self.model.appendRow(item)
            widget = DashboardWidget(item, self)
            self.tabWidget.addTab(widget, "New Dashboard")

            item.tab = widget

        pass
    def closeTabFromTabWidget(self, i):
        widget = self.tabWidget.widget(i)
        modelIndex = widget.model.index()
        self.navListView.model().removeRow(modelIndex.row())
        self.tabWidget.removeTab(self.tabWidget.indexOf(widget.model.tab))
```

```
        for i in range(self.model.rowCount()):
            el=self.model.itemFromIndex(self.model.index(i, 0))
            if(el.model_name =='D'):
                el.tab.Settings_D.update_all_charts(self.model)
                ind=el.tab.Settings_D.Setting_SH.index(widget.stackedWidget.widget(0))

                el.tab.Settings_D.Setting_SH[ind].chartView_D.setParent(None)
                el.tab.Settings_D.Setting_SH.remove(el.tab.Settings_D.Setting_SH[ind])

    def createActions(self):
        self.newChartAction = QAction("New Chart", self,  icon = QIcon('icons/chart.svg'))
        self.newChartAction.triggered.connect(self.newChart)

        self.newTableAction = QAction("New Table", self, icon = QIcon('icons/table.svg'))
        self.newTableAction.triggered.connect(self.newTable)

        self.newDashboardAction = QAction("New Dashboard", self,   icon =
QIcon('icons/dashboard.svg'))
        self.newDashboardAction.triggered.connect(self.newDashboard)

    def setActiveTab(self):
        selectedModel = self.model.itemFromIndex(self.navListView.selectedIndexes()[0])
        self.tabWidget.setCurrentWidget(selectedModel.tab)

if  __name__== "__main__":
    import sys
    app = QApplication(sys.argv)
    window = MainWindowV2()
    window.setWindowTitle("Приложение для визуализации динамических данных")
    window.resize(900, 600)
    window.show()
    sys.exit(app.exec_())
```