Санкт-Петербургский государственный университет

**Кафедра технологии программирования**

**Бурдейный Михаил Анатольевич**

**Выпускная квалификационная работа бакалавра**

**Обработка кластерных данных в цифровой экономике**

Направление 01.03.02

Прикладная математика и информатика

Научный руководитель,
старший преподаватель
Максимов А.Ю.

Санкт-Петербург

2018

[Введение 3](#_Toc515016897)

[Постановка задачи 10](#_Toc515016898)

[Обзор литературы 11](#_Toc515016899)

[Глава 1. Кластеризация 16](#_Toc515016900)

[1.1. Структура программы 16](#_Toc515016901)

[1.2. Иерархический метод кластеризации 16](#_Toc515016902)

[1.3. K-средних метод кластеризации 21](#_Toc515016903)

[1.4. Алгоритм минимального остовного дерева 27](#_Toc515016904)

[Глава 2. Прогнозирование 35](#_Toc515016905)

[2.1 Прогнозирование ситуации 35](#_Toc515016906)

[2.2. Графическая составляющая приложения 38](#_Toc515016907)

[2.3. Анализ модуля предсказаний 42](#_Toc515016908)

[2.4. Инструкция для пользователей (скриншоты) 43](#_Toc515016909)

[Вывод 48](#_Toc515016910)

[Список литературы 50](#_Toc515016911)

[Приложение 53](#_Toc515016912)

#  Введение

В связи с экспоненциальным развитием, цели и задачи развития цифровой экономики в Российской Федерации определены до 2024 года в рамках пяти базовых направлений:

- нормативное регулирование;

- кадры и образование;

-формирование исследовательских компетенций и технических заделов;

 -информационная инфраструктура;

- информационная безопасность.

Основой развития цифровой экономики является работа с большими данными. К появлению концепции «больших данных» (big data) привело увеличившееся использование цифровых устройств, интернета. Потоки данных постоянно возрастают (уже терабайты и петабайты), передаются в реальном времени, обрабатываются и используются для принятий решений.

Развитие цифровой экономики позволяет объединять усилия для инвестирования, поиска сотрудников, партнеров, ресурсов и рынков сбыта способствует развитию новых моделей бизнеса, обеспечивает возможность коммуникаций, обмена идеями знаниями и опытом. Цифровые технологии могут играть ведущую роль в реализации инновационных идей, в различных сферах, в том числе в социальной сфере.

Возможности, создаваемые big data и современными системами их поиска, для развития различных отраслей, науки и менеджмента, характеризуются как беспрецедентные, и способствуют к переходу на новый уровень анализа и управления экономическими процессами, как на уровне отдельных отраслей, предприятий. регионов, так и на макроуровне, совершенствованию моделирования и прогнозирования социально-экономического развития.

Интеллектуальный анализ данных — кластеризация в Data Mining приобретает ценность тогда, когда она выступает одним из этапов анализа данных, построения законченного аналитического решения. Аналитику часто легче выделить группы схожих объектов, изучить их особенности и построить для каждой группы отдельную модель, чем создавать одну общую модель для всех данных. Таким приемом постоянно пользуются в маркетинге, выделяя группы клиентов, покупателей, товаров и разрабатывая для каждой из них отдельную стратегию.

Обширной информационной ёмкостью отличается область исследований дорожно-транспортных происшествий (ДТП)**.** Она требует привлечения и анализа больших массивов данных, разнообразных математических, алгоритмических и программных средств.

Высокий уровень аварийности на автомобильном транспорте, постоянно растущее число погибших и раненых в (ДТП), является одной из острейших социально-экономических проблем для всех стран мира, отражаясь на экономике государств.

В соответствии с информацией Всемирной организации здравоохранения (ВОЗ) общее количество погибших на автомобильных дорогах мира более 1 млн. 200 тысяч в год, к 2030 году составит -3,6 млн. в год

В России число погибших на 10 тысяч транспортных средств в 3–5 раз превышает аналогичные показатели Европейских зарубежных стран, число погибших на 100 тысяч населения выше в 1,5–2 раза.

Задача снижения уровня аварийности на автомобильном транспорте может более эффективно решаться с развитием цифровой экономики (внедрением систем видео фиксации, решением многочисленных частных задач исследования ДТП, кластерного подхода, обнаружением всех потребителей информации, сопровождающей эти задачи, минимизированием общих затрат при существенном повышении качества мероприятий).

Количество статистических данных о ДТП, которое необходимо учесть и обработать, достаточно велико и требует широкого применения современных средств вычислительной техники и программного обеспечения.

 Эти факторы побудили меня к созданию программного продукта для решения задачи кластерного анализа ДТП.

Указанный программный продукт пользовательское приложение «Кластеризация», помимо анализа ДТП, может применяться в очень широком спектре: в маркетинге, медицине, археологии, психологии, химии, биологии, психологии государственном управлении, филологии, социологии, геологии, антропологии и других.

В информатике для «интеллектуальной» группировки результатов при запросах поиска файлов, веб-сайтов и других объектов, целесообразно использовать кластеризацию результатов поиска, дав пользователю возможность быстрой навигации, выборки заведомо более релевантного подмножества и исключения заведомо менее релевантного — что значительно повышает юзабилити интерфейса по сравнению с выводом в виде простого сортированного по релевантности списка. В настоящее время разработаны и применяются.

 Clusty — кластеризующая поисковая машина компании Vivísimo.

Nigma — российская поисковая система с автоматической кластеризацией результатов.

Quintura — визуальная кластеризация в виде облака ключевых слов.

Сегментация изображений — разбиение цифрового изображения на отдельные области с целью обнаружения границ распознавания объектов.

Кластерный анализ – является многомерной процедурой, которая производит сбор входных данных, содержащих различную информацию о выборке объектов, после чего совершает упорядочивание объектов в максимально возможные однородные группы. Основной задачей кластеризации является задача статистической обработки, а также может относиться к обширному классу задач обучения без учителя.

В дальнейшем появились термины - синонимы, например, автоматическая классификация и ботриология.

Основные задачи кластерного анализа:

исследование концептуальных схем, используемых для группирования объектов;

порождение гипотез на основе исследования данных;

разработка типологии или классификации;

проверка гипотез или исследования для определения, присутствуют ли в имеющихся данных типы (группы), выделенные тем или иным способом,

Вне зависимости от предмета изучения и методов кластерного анализа, предполагаются следующие этапы:

отбор выборки для кластеризации. Кластеризуются исключительно количественные данные;

нахождение множества переменных, при помощи которых, можно оценить объекты в выборке;

вычисление значений той или иной меры сходства (или различия) между объектами;

создание групп сходных объектов, путем применения метода кластерного анализа;

проверка достоверности результатов кластерного решения.

В практике встречается описание двух фундаментальных требований, которые предъявляются к данным — однородность и полнота. Для однородности необходимо, чтобы все кластеризуемые сущности были непосредственно одной природы, и были описаны схожим набором характеристик. В том случае, если кластерному анализу предшествует факторный анализ, то сама выборка не нуждается в «ремонте», т. е все изложенные требования будут выполнены автоматически, самой процедурой факторного моделирования (это является еще одним достоинством — z-стандартизация без отрицательных последствий для выборки; если её проводить непосредственно для кластерного анализа, то она может повлечь за собой существенной уменьшение чёткости разделения групп). В противном случае выборку необходимо корректировать.

Основные цели кластеризации:

понять данные при помощи выявления кластерной структуры. Разбить выборки на группы подобных объектов, что позволит упростить дальнейшую обработку данных и принятие решений, путем применения к каждому кластеру своего метода анализа (стратегия «разделяй и властвуй»);

сжатие данных. Если исходная выборка слишком большая, можно сократить её, отделив от каждого кластера, одного, наиболее типичного представителя;

обнаружение новизны. Выделяются объекты, которые не удаётся присоединить ни к одному из кластеров; их принято называть нетипичными.

Методы кластеризации.

Общепринятой классификации методов кластеризации, как правило, не существует, но мы можем выделить ряд групп подходов

1. *Вероятностный подход*. Предполагается, что каждый объект будет, относится к одному из k классов. Некоторые ученые считают, что данная группа вовсе не относится к кластеризации и называют ее «дискриминация», то есть выбор отнесения объектов к одной из известных групп (обучающих выборок).
	* K-средних
	* К-медиан
	* EM-алгоритм
	* Алгоритмы семейства FOREL
	* Дискриминантный анализ
2. Подходы на основе использования искусственного интеллекта: весьма условная группа, так как существует очень много методов, и они весьма различны.
	* Метод нечеткой кластеризации C-средних (C-means)
	* Нейронная сеть Кохонена
	* Генетический алгоритм
3. Логический подход. Построение дендрограммы при помощи дерева решений.
4. Теоретико-графовый подход.
	* Графовые алгоритмы кластеризации
5. Иерархический подход. Предполагает собой наличие неких вложенных групп (кластеров различного порядка). В свою очередь, алгоритмы подразделяют на агломеративные и дивизивные. По количеству признаков иногда выделяют монотетические и политетические методы классификации.
	* Иерархическая дивизивная кластеризация или таксономия. Задачи кластеризации рассматриваются в количественной таксономии.
6. Другие методы.
	* Статистические алгоритмы кластеризации
	* Ансамбль кластеризаторов
	* Алгоритмы семейства KRAB
	* Алгоритм, основанный на методе просеивания
7. DBSCAN и др.

Мы применили данные алгоритмы кластеризации, в очень важной и значимой задаче, анализ ДТП. В настоящее время анализ и мониторинг дорожно-транспортной аварийности связан, как правило, с «узкими» местами и местами концентрации ДТП. Однако, «недостаточно установить, что происшествие произошло из-за неверного действия водителя. Мы должны также спросить, почему было выполнено неправильное действие. Можно представить, что большая часть объяснений неправильных действий в дорожном движении заключается в том, что система дорожного движения в данных ситуациях предъявляет высокие требования к работоспособности человека.

Если система будет слишком сложной, то даже наиболее хорошо оснащённые участники дорожного движения будут время от времени совершать фатальные ошибки». Поэтому необходимо анализировать и оценивать уровень обеспечения БДД не только в конкретных местах, но и в их окрестностях.

Объектом нашего ислледования, является количество ДТП, в совокупности, с количеством пострадавших, за определенный промежуток времени, по нескольким регионам РФ.

Задача формирования перечня мероприятий владельцев автомобильных дорог по повышению БДД с целью выбора объектов для реализации мероприятий с наибольшей эффективностью может решаться только при проведении анализа и постоянного мониторинга.

# Постановка задачи

Разработать пользовательское приложение “Кластеризация”, при помощи языка программирования Python, с использованием сторонних библиотек.

Приложение должно преобразовывать входные данные (.xsl, .xslx, .csv файлы) в вид необходимый для применения алгоритмов кластеризации и прогнозирования, и выполнять необходимые действия над ними. Под действиями подразумевается применения следующих методов кластеризации:

* Иерархический
* K-средних
* Минимального покрывающего дерева

Прогнозирование при помощи библиотеки от известной компании Facebook, fbprophet.

Выходным результатом будут графики распределения расстояния между данными и дендограмма, для иерархического метода; трехмерный график объектов, входящих в кластер для метода k-средних; и минимальное остовое и результирующее дерево для алгоритма минимального покрывающего дерева. График тренда, годовой и недельной сезонности по которым будет возможно увидеть прогнозирование ситуации. А также таблицы данных для всех методов кластеризации

Кластерный анализ проведён для 85 областей РФ, за срок с 2007 г — по 2017 г., по 3 признакам: количество ДТП (1-й признак), количество погибших (2-й признак), количество раненых (3-й признак). Предварительный анализ входных данных, позволил выдвинуть гипотезу, что все территории возможно разделяются на ***5 кластеров***.

# Обзор литературы

В данном проекте использовалось большое количество, как отечественной литературы, так и зарубежной.

Для изучения вопроса о применении методов кластерного анализа в цифровой экономике с целью решения поставленной задачи, было проанализировано множество алгоритмов кластеризации.

Решение задачи кластеризации является неоднозначным. В - первых, не существует всеми принятого критерия качества кластеризации. Существует множество логически или эмпирически выведенных критериев качества, также существует множество алгоритмов, которые не имеют четко выраженного критерия качества, но осуществляющих достаточно разумную кластеризацию.

Во-вторых, число кластеров, как правило, является неизвестным заранее и устанавливается определенным критерием, отличающимся у различных конкретных алгоритмов. В-третьих, результат кластеризации существенно зависит от выбора метрики 𝜌, выбор которой также субъективен и определяется экспертом [3, 18]. Поскольку выбор метрики зачастую является абсолютно субъективным, то для решения задачи кластеризации создаются новые, всё более сложные, алгоритмы и улучшаются старые.

Одним из широко применяемых алгоритмов для кластеризации является алгоритм k-средних [5, 8, 15, 24, 27, 29]. Алгоритм k-средних производит разбиение входной выборки на 𝑘 кластеров. Действие алгоритма таково, что он стремится минимизировать среднеквадратичное отклонение точек кластера от его центра. Основная мысль алгоритма заключается в том, что на каждом шаге пере вычисляется центр масс каждого кластера, полученного на предыдущем шаге, а затем выборка пере разбивается на кластеры вновь в соответствии с тем, какой из новых центров кластеров оказался ближе по выбранной метрике. Алгоритм завершается, когда на каком-либо шаге не происходит изменения кластеров.

Достоинствами метода являются его простота и быстрота использования*,* его понятность и прозрачность.

Недостатками алгоритма являются: чувствительность к выбросам, необходимость задавать количество кластеров, необходимость сканировать всю выборку для определения положения каждого кластера. Из-за медленной работы алгоритма на больших выборках данных, а также простоты идеи и неплохих результатов для большинства выборок, предлагается множество способов ускорить или улучшить алгоритм k-средних.

Так, в работе [15] для ускорения работы алгоритма k- средних, предложено использование неравенства треугольника. Ускоренный алгоритм избегает излишних вычислений расстояния с помощью сохранения «верхних» и «нижних» границ для дистанций между точками и центрами, для расчета которых двумя различными методами применяется неравенство треугольника. Ускоренный алгоритм выдает те же результаты, что и обычный, но при этом количество вычислений дистанций сокращается в десятки раз.

 Еще один вариант ускорения алгоритма путем сокращения лишних вычислений предлагают авторы статьи [28]: после каждой итерации алгоритма необходимо разделять кластеры на те, которые изменили своё положение, так называемые «активные», и «статические» — оставшиеся на месте; и продолжать дальнейшие расчеты для активных кластеров. При этом для корректировки списков сохраняется минимальное значение расстояний от центров кластера до точек, при достижении которых кластер снова становится «активным».

 В статье [24] предлагается «алгоритм фильтрации». Он так же основан на хранении многомерных данных в структуре kd- деревьев, а отличается тем, что выбор кластера, которому должен принадлежать лист, осуществляется построением гиперплоскости между претендентами.

 В диссертации [23] был описан «быстрый жадный алгоритм k-средних». Предположение, используемое в алгоритме, такое же, как в обычном жадном (глобальном) алгоритме [29], оно заключается в том, что глобальный оптимум может быть достигнут при запуске алгоритма с (𝑘 − 1) центрами, и k-тым центром, достраиваемым автоматически на подходящую позицию.

Результат работы алгоритма k-средних очень сильно зависит от начального распределения кластеров. В работе [11] рассматривается метод k-means++, в котором начальное распределение центров кластеров задается особым образом, опираясь на вероятности, рассчитанные по кратчайшим расстояниям от точек до выбранных центров. В работе [5] представлен алгоритм квантования цветов, основанный на методе k-средних и дополненный алгоритмом гравитационного поиска [14]. После стандартного расчета расстояний по метрике, рассчитываются «массы» кластеров и «силы», действующие на них. Таким образом, в работе достигается улучшение значений заданной целевой функции*.*

Работа [31] описывает построение дерева с использованием двух алгоритмов: иерархического и приближенного.

Иерархический алгоритм строит дерево результатов, на каждом следующем уровне разбивая имеющиеся кластеры на количество, называемое «коэффициентом разветвления», при этом каждый из полученных кластеров рассчитывается независимо от других. В приближенном алгоритме изначально каждое дерево развернуто до листьев, затем итерационно создается очередь приоритетных узлов до достижения конкретного числа открытых путей по дереву. При этом увеличение количества деревьев не приводит к значительному увеличению затрачиваемого времени.

Определение оптимального количества кластеров во входной выборке является одной из сложнейших проблем в кластерном анализе [10, 11].

Также существует множество других алгоритмов кластеризации, основанных на различных идеях о поиске внутренней структуры данных. Можно выделить несколько больших групп алгоритмов по способу кластеризации [6] и [21]: разделяющие методы, иерархические методы, плотностные методы и сеточные методы.

Авторы работы [1] и научной публикации [33] объясняют алгоритм в основе Python [1] и библиотеки Prophet [33], используемых для решения поставленной задачи.

 Авторы работы [32] указывают, что цифровая экономика явно является путеводителем экономических и социальных изменений в настоящее время, привлечения инвестиций, повышения производительности.

 «Цифровая экономика обусловлена конвергенцией информации, вычислений и коммуникаций, которые пришли из Интернета (широкомасштабный рост электронной коммерции, новых конкурентных стратегий и изменений в бизнес-процессы и организационной структуры) это позволяет новые сетевые формы деятельности, которые основаны не на иерархии, а на отношениях. Интернет-технологии являются прямым драйвером изменений в коммерции и в структуре данных и операциях.» [32].

 Эти изменения уже хорошо документированы в случае исследований, но данных очень мало, хотя оценки роста электронной торговли широко цитируется. Есть также исследования, предполагающие, что Интернет позволяет сетевые формы деятельности, которые охватывают организационные границы (как в экстрасети) или представляет новые, более открытые формы организации (иногда называемые «сообществами»). Мы видим примеры вокруг нас о том, как эти новые технологии используются в научных исследованиях, здравоохранении, образовании и управлении. Мы ежедневно слышим о путях, в которых технология бросает вызов традиционным законам, политике и институтам. Хотя мы знаем, что эта технология приносит значимые экономические, правовые, социальные, этические, политические и культурные изменения вместе с ним, федеральное правительство спонсировало социальные научные исследования в этой области [32].

Как утверждает ряд экспертов, в настоящее время для экономического агента становится важным не сам факт обладания каким-либо ресурсом, а наличие данных об этом ресурсе и возможность их использовать с целью планирования своей деятельности [4].

Данные полученные с таких устройств, как цифровые девайсы, смартфоны, интернет вещей позволяют создавать цифровые модели потребителей, технологических процессов, что приводит к экономии ресурсов, оптимизации систем закупок, оптимизации использования финансов и т.д. [7]

Одним из первопроходцев в научной сфере статистического анализа являлся программный продукт **S-tatistica.**

**Statistica —**[программный пакет](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D0%BA%D0%B5%D1%82_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D1%8B%D1%85_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC)для статистического анализа, разработанный компанией StatSoft, реализующий функции [анализа данных](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), [управления данными](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%BC%D0%B8), добычи данных, [визуализации данных](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)с привлечением [статистических методов](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0).

Мы добавили в наш продукт последние достижения, в области математического проектирования, и прогнозирования, что позволило получить продукт, выполняющий гораздо больший спектр задач, с большей производительностью, и меньшей ресурсоемкостью.

# Глава 1. Кластеризация

## Структура программы

Данная программа имеет следующую структуру:

* **Data.csv**
* **Data1.csv**
* **Foo.png**
* **hierarchyCls.py**
* **HierarchyTable.py**
* **KMeansCls.py**
* **KMeansTable.py**
* **Main.py**
* **MstCls.py**
* **Predict.py**
* **shared.py**

В файлах **Data.csv** и **Data1.csv** содержатся входные данные для программы. В **foo.png** содержится результат прогнозирования ситуации. Модули **hierarchyCls.py|hierarchyTable.py**;**KMeansCls.py| KMeansTable.py**; и **MstCls** отвечают за кластеризацию иерархическим, K-средних и Mst методами соответственно, а также построение и отображение таблиц данных.

Файл **Predict.py** отвечает за построение прогноза данных, а **shared.py** для получения и нормализации входных данных. **Main.py** является главным файлом программы, и содержит в себе создание графической части приложения, а так же объединяет и синхронизирует работу всех модулей программы.

## Иерархический метод кластеризации

Иерархическая кластеризация (также графовые алгоритмы кластеризации) — совокупность алгоритмов упорядочивания данных, визуализация которых обеспечивается с помощью [графов](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84_%28%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29).

Алгоритмы упорядочивания данных указанного типа исходят из того, что некое множество объектов характеризуется определённой степенью связности. Предполагается наличие вложенных групп (кластеров различного порядка). Алгоритмы, в свою очередь, подразделяются на агломеративные (объединительные) и дивизивные (разделяющие). По количеству признаков иногда выделяют монотетические и политетические методы классификации. Как и большинство визуальных способов представления зависимостей графы быстро теряют наглядность при увеличении числа объектов.

Под дендрограммой обычно понимается [дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%29), то есть граф без циклов, построенный по матрице мер близости. Дендрограмма позволяет изобразить взаимные связи между объектами из заданного множества. Для создания дендрограммы требуется [матрица сходства](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BC%D0%B5%D1%80_%D0%BA%D0%BE%D0%BD%D0%B2%D0%B5%D1%80%D0%B3%D0%B5%D0%BD%D1%86%D0%B8%D0%B8#%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BE%D1%82%D0%BD%D0%BE%D1%81%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85_%D1%81%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D1%85_%D0%BC%D0%B5%D1%80_%D0%BA%D0%BE%D0%BD%D0%B2%D0%B5%D1%80%D0%B) (или [различия](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D1%80%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B9)), которая определяет уровень сходства между парами объектов. Чаще используются агломеративные методы.

Построение дендограммы происходит с использованием следующей формулы.



В контексте программы, построение гистограммы расстояний (Рис.1) происходит следующим образом:

plt.hist(dist, 500, color='green', alpha= 0.5), где dist – матрица попарных расстояний между объектами. В качестве меры расстояния было взято обычное расстояние Эвклида.

Построение дендограммы (Рис.1):

Z = hierarchy.linkage(dist, method='average') level = .25

hierarchy.dendrogram(Z, labels=labels, color\_threshold=level, leaf\_font\_size=5, count\_sort=True), где Z -  полученное в результате иерархической кластеризации дерево полного разбиения, при помощи функции linkage, передав ей в параметрах полученные ранее данные и указав метод попарного среднего в качестве правила объединения кластеров. А level – пороговый уровень отрисовки, равный 0.25

Для построения графика данных(Рис.2), из гистограммы получаются следующие данные:

 counts, bins, bars = plt.hist(dist, 500, color='green', alpha=0.5)



Рисунок 1. Графики иерархической кластеризации



Рисунок 2. Таблица иерархической кластеризации

Основываясь на результатах дендограммы, можем сделать вывод, что данный метод выделил 5 кластеров входных данных, т.е упорядочил их по 5 группам. В каждом кластере содержатся выделенные объекты.

**1-й кластер:**

1. **Москва**
2. **Санкт-Петербург**

**2-й кластер:**

1. **Московская область**
2. **Краснодарский край**

**3-й кластер:**

1. **Ростовская область**
2. **Башкортостан**
3. **Белгородская область**
4. **Татарстан**
5. **Челябинская область**

**4-й кластер:**

1. **Омская область**
2. **Тюменская область**
3. **Приморский край**
4. **Новосибирская область**
5. **Кемеровская область**
6. **Алтайский край**
7. **Оренбургская область**
8. **Тульская область**
9. **Волгоградская область**
10. **Владимирская область**
11. **Крым**
12. **Свердловская область**
13. **Дагестан**
14. **Иркутская область**
15. **Пермский край**
16. **Саратовская область**
17. **Красноярский**
18. **Самарская область**
19. **Ставропольский**
20. **Воронежская область**
21. **Ленинградская область**

**5-й кластер:**

1. **Амурская область**
2. **Брянская область**
3. **Бурятия**
4. **Ямало-Ненецкая АО**
5. **Севастополь**
6. **Мурманская область**
7. **Костромская область**
8. **Карачаево-Черкесская область**
9. **Чеченская**
10. **Магаданская область**
11. **Алтай**
12. **Еврейская АО**
13. **Чукотский АО**
14. **Ненецкий АО**
15. **Новгородская**
16. **Калининградская**
17. **Мордовия**
18. **Псковская**
19. **Якутия**
20. **Смоленская**
21. **Астраханская**
22. **Коми**
23. **Чувашия**
24. **Белгородская**
25. **Тамбовская**
26. **Волгоградская**
27. **Архангельская**
28. **Ивановская**
29. **Кировская**
30. **Калужская**
31. **Ульяновская**
32. **Курская**
33. **Ханты-Мансийская**
34. **Рязанская**
35. **Тверская**
36. **Пензенская**
37. **Удмуртская**
38. **Хабаровский край**
39. **Ярославская**
40. **Забайкальский край**
41. **Липецкая**
42. **Хакасия**
43. **Карелия**
44. **Тыва**
45. **Адыгея**
46. **Кабардино-Балкария**
47. **Сахалинская**
48. **Томская**
49. **Марий Эл**
50. **Орловская**
51. **Северная Осетия**
52. **Камчатский**
53. **Калмыкия**
54. **Ямало-Ненецкая АО**
55. **Севастополь**
56. **Мурманская**
57. **Костромская**
58. **Карачаево-Черкесская**
59. **Чеченская**
60. **Магаданская**
61. **Алтай**
62. **Еврейская АО**
63. **Чукотский АО**
64. **Ненецкий АО**

## K-средних метод кластеризации

Метод k-средних ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) k-means) — наиболее популярный метод [кластеризации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F). Был изобретён в 1950-х годах математиком [Гуго Штейнгаузом](https://ru.wikipedia.org/wiki/%D0%A8%D1%82%D0%B5%D0%B9%D0%BD%D0%B3%D0%B0%D1%83%D0%B7%2C_%D0%93%D1%83%D0%B3%D0%BE) и почти одновременно Стюартом Ллойдом. Особую популярность приобрёл после работы Маккуина.

Действие алгоритма таково, что он стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров:



где k— число кластеров, Si — полученные кластеры, i = 1, 2,…, k и µi — центры масс векторов xj∈ Si.

По аналогии с [методом главных компонент](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B3%D0%BB%D0%B0%D0%B2%D0%BD%D1%8B%D1%85_%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82) центры кластеров называются также главными точками, а сам метод называется методом главных точек и включается в общую теорию главных объектов, обеспечивающих наилучшую [аппроксимацию](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D1%80%D0%BE%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) данных.

Алгоритм представляет собой версию [EM-алгоритма](https://ru.wikipedia.org/wiki/EM-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), применяемого также для разделения смеси  [гауссиан](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D1%83%D1%81%D1%81%D0%B8%D0%B0%D0%BD%D0%B0). Он разбивает [множество](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) элементов пространства на заранее известное число кластеров k.

Основная идея заключается в том, что на каждой итерации пере вычисляется [центр масс](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80_%D0%BC%D0%B0%D1%81%D1%81) для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике.

Алгоритм завершается, когда на какой-то итерации не происходит изменения внутрикластерного расстояния. Это происходит за конечное число итераций, так как количество возможных разбиений конечного множества конечно, а на каждом шаге суммарное квадратичное отклонение V уменьшается, поэтому зацикливание невозможно.

Проблемы:

* Не гарантируется достижение глобального минимума суммарного квадратичного отклонения V, а только одного из локальных минимумов.
* Результат зависит от выбора исходных центров кластеров, их оптимальный выбор неизвестен.
* Число кластеров надо знать заранее.

В алгоритмах [глубокого обучения](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D1%83%D0%B1%D0%BE%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) метод k-средних иногда применяют не по прямому назначению (классификация разбивкой на кластеры), а для создания так называемых фильтров (ядер свёртки, словарей). Например, для распознавания изображений в алгоритм k-средних подают небольшие случайные кусочки изображений обучающей выборки, допустим, размером 16х16 в виде линейного вектора, каждый элемент которого кодирует яркость своей точки. Количество кластеров k задается большим, например, 256. Обученный метод k-средних при определенных условиях вырабатывает при этом центры кластеров (центроиды), которые представляют собой удобные базисы, на которые можно разложить любое входное изображение. Такие "обученные" центроиды в дальнейшем используют в качестве фильтров, например для [свёрточной нейронной сети](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%B5%D1%80%D1%82%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C) в качестве ядер свёртки или других аналогичных систем машинного зрения. Таким образом осуществляется обучение без учителя при помощи метода k-средних.

Данный алгоритм разбивает исходное множество объектов на k кластеров так, что средние в кластере (для всех переменных) максимально возможно отличаются друг от друга.
Выбор числа k может базироваться на результатах предшествующих исследований, теоретических соображениях или интуиции. Если нет предположений относительно этого числа, рекомендуют создать 2 кластера, затем 3, 4, 5 и т.д., сравнивая полученные результаты.

Работа алгоритма делится на несколько этапов:

* случайно выбрать k точек, являющихся начальными «центрами масс» кластеров (число k выбирается исследователем);
* отнести каждый объект к кластеру с ближайшим «центром масс»;
* пересчитать «центры масс» кластеров согласно их текущему составу;
* если критерий остановки алгоритма не удовлетворен, вернуться к п. 2.

В качестве критерия остановки работы алгоритма обычно выбирают одно из двух условий:

* «центром масс» кластеров стабилизировались, т.е. все наблюдения принадлежат кластеру, которому принадлежали до текущей итерации;
* число итераций равно максимальному числу итераций.

В контексте программы, сам алгоритм реализован в библиотеке [scipy](http://docs.scipy.org/). Функция kmeans(obs, k\_or\_guess, iter=20, thresh=1e-05)

принимает на вход набор данных и количество искомых кластеров.

centroids = vq.kmeans(numpy.array(data), 7, iter=200)[0] -  получение координат центров найденных кластеров (т. н. центроидов), с помощью которых разобьем исходное множество образцов на кластеры.

Теперь необходимо сопоставить каждый объект исходного набора ближайшему к нему центру кластера. Проще всего сделать это с помощью функции cdist(), которая рассчитывает и возвращает расстояния между всеми элементами двух списков.

res = [[] for i in xrange(len(centroids))]

 d = cdist(numpy.array(data), centroids, 'euclidean')

 for i, l in enumerate(d):

 res[l.tolist().index(l.min())].append((labels[i], data[i]))

 kmeans\_draw(res, plt, fig)

В результате у нас появился список кластеров, каждый из которых содержит данные о входящих в него объектах, включая его имя и характеристики.

Поскольку у каждого объекта всего 3 характеристики мы используем их как координаты в трёхмерном пространстве (если бы их было больше - пришлось бы применять так называемые алгоритмы многомерного шкалирования для вывода точек на плоскость). Для отображения воспользуемся библиотечкой [matplotlib](http://matplotlib.sourceforge.net/), а именно её тулкитом для трёхмерных графиков.

colors = deque(['r', 'g', 'b', 'c', 'm', 'y', 'k'])

 ax = Axes3D(fig)

 for cluster in clusters:

 color = colors.popleft()

 for name, coord in cluster:

 x, y, z = coord

 names.append(name)

 X.append(x)

 Y.append(y)

 Z.append(z)

 ax.plot3D([x], [y], [z], marker='o', c=color)

 ax.set\_xlabel(u'ДТП')

 ax.set\_ylabel(u'Погибло')

 ax.set\_zlabel(u'Ранено')

Данная функция перебирает все полученные кластеры и рисует каждый входящий в него объект в виде цветной точки на трёхмерном графике(Рис.3).

Функция kmeans\_export возвращает нам имя записи, а также ее координаты на графике:

return names, X, Y, Z

Эти данные мы можем использовать для построения таблицы (Рис.4).



Рисунок 3. График метода K-средних



Рисунок 4. Таблица метода K-средних

данный метод выделил 7 кластеров входных данных, т.е упорядочил их по 7 группам.

В каждом кластере содержатся выделенные объекты.

**1-й кластер:**

1. **Москва**
2. **Санкт-Петербург**

**2-й кластер:**

1. **Московская область**
2. **Краснодарский край**

**3-й кластер:**

1. **Ростовская область**
2. **Башкортостан**
3. **Белгородская область**
4. **Татарстан**
5. **Челябинская**

**4-й кластер:**

1. **Тульская**
2. **Волгоградская**
3. **Владимирская**
4. **Крым**
5. **Свердловская**
6. **Дагестан**
7. **Иркутская**
8. **Пермский край**
9. **Саратовская**
10. **Красноярский**
11. **Самарская**
12. **Ставропольский**
13. **Воронежская**
14. **Ленинградская**

**5-й кластер:**

1. **Амурская**
2. **Брянская**
3. **Бурятия**
4. **Новгородская**
5. **Калининградская**
6. **Мордовия**
7. **Псковская**
8. **Якутия**
9. **Смоленская**
10. **Астраханская**
11. **Коми**
12. **Чувашия**
13. **Белгородская**
14. **Тамбовская**
15. **Волгоградская**
16. **Архангельская**
17. **Ивановская**
18. **Северная Осетия**
19. **Камчатский**
20. **Калмыкия**
21. **Ямало-Ненецкая АО**
22. **Севастополь**
23. **Мурманская**
24. **Костромская**
25. **Карачаево-Черкесская**
26. **Чеченская**
27. **Магаданская**
28. **Алтай**
29. **Еврейская АО**
30. **Чукотский АО**
31. **Ненецкий АО**

**6-й кластер:**

1. **Омская область**
2. **Тюменская**
3. **Приморский край**
4. **Новосибирская**
5. **Кемеровская**
6. **Алтайский край**
7. **Оренбургская**

**7-й кластер:**

1. **Кировская**
2. **Калужская**
3. **Цльяновская**
4. **Курская**
5. **Ханты-Манстйская**
6. **Рязанская**
7. **Тверская**
8. **Пензенская**
9. **Удмуртская**
10. **Хаабаровский край**
11. **Ярославская**
12. **Забайкальский край**
13. **Липецкая**
14. **Хакасия**
15. **Карелия**
16. **Тыва**
17. **Адыгея**
18. **Кабардино-Балкария**
19. **Сахалинская**
20. **Томская**
21. **Марий Эл**
22. **Орловская**

## Алгоритм минимального остовного дерева

Минимальное остовное дерево (или минимальное покрывающее дерево) в связанном взвешенном [неориентированном графе](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) — это [остовное дерево](https://ru.wikipedia.org/wiki/%D0%9E%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) этого графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов входящих в него рёбер.

Обширный класс алгоритмов кластеризации основан на представлении выборки в виде графа. Вершинам графа соответствуют объекты выборки, а рёбрам — попарные расстояния между объектами.

Достоинством графовых алгоритмов кластеризации является наглядность, относительная простота реализации и возможность вносить различные усовершенствования, опираясь на простые геометрические соображения.

Алгоритм минимального покрывающего дерева (MST) строит граф из N-1 рёбер так, чтобы они соединяли все N точек и обладали минимальной суммарной длиной. Такой граф называется кратчайшим незамкнутым путём, минимальным покрывающим деревом или каркасом графа.

Описание работы алгоритма.
Найти пару точек с наименьшим расстоянием (весом ребра) и соединить их ребром,

* пока в выборке остаются изолированные точки:
* найти изолированную точку, ближайшую к некоторой неизолированной;
* соединить эти две точки ребром;
* удалить K-1 самых длинных рѐбер;

На последнем шаге алгоритм удаляет из полученного графа K-1 самых длинных рёбер, что приводит к распаду графа на отдельные связанные компоненты, которые и будут искомыми кластерами.

Общеизвестны два недостатка этого алгоритма:

ограниченная применимость. Алгоритм наиболее подходит для выделения кластеров типа сгущений или лент. Наличие разреженного фона или «узких перемычек» между кластерами приводит к неадекватным результатам;

высокая трудоёмкость — для построения кратчайшего незамкнутого пути требуется O(N^3) операций.

Задача о нахождении минимального остовного дерева часто встречается в подобной постановке: допустим, есть n городов, которые необходимо соединить дорогами, так, чтобы можно было добраться из любого города в любой другой (напрямую или через другие города). Разрешается строить дороги между заданными парами городов и известна стоимость строительства каждой такой дороги. Требуется решить, какие именно дороги нужно строить, чтобы минимизировать общую стоимость строительства.

Эта задача может быть сформулирована в терминах теории графов как задача о нахождении минимального остовного дерева в графе, вершины которого представляют города, рёбра — это пары городов, между которыми можно проложить прямую дорогу, а вес ребра равен стоимости строительства соответствующей дороги.

В контексте программы, для работы с графами будет использоваться [NetworkX](http://networkx.github.io/) — библиотека для создания и манипуляции графами. Кроме этого, она предоставляет [готовую реализацию алгоритма нахождения минимального остовного дерева](http://networkx.github.io/documentation/latest/reference/algorithms.mst.html?highlight=mst#networkx.algorithms.mst) и удобное API для визуализации результатов.

Попробуем построить граф, представляющий наш исходный набор данных. Каждая вершина этого графа будет обозначать отдельный объект исходной выборки и будет связана со всеми остальными вершинами посредством взвешенных рёбер. Вес каждого ребра будет равен расстоянию близости между объектами.

s = nx.Graph() # исходный граф

 s.add\_nodes\_from(labels)

 r = s.copy() # результат кластеризации

 dq = deque(dist)

 len\_x = len(labels)

 for x in xrange(len\_x - 1):

 for y in xrange(x + 1, len\_x):

 s.add\_edge(labels[x], labels[y], weight=dq.popleft())

В переменной s содержится исходный граф нашего набора данных. Следующим шагом построим минимальное покрывающее дерево полученного графа и построим таблицу весов рёбер в нём (дабы выбрать пороговый уровень веса) (Рис.5).

mst = nx.minimum\_spanning\_tree(s)

 counts, bins, bars = plt.hist([edge[2]['weight'] for edge in mst.edges\_iter(data=True)], 100, color='red', alpha=0.3)

Теперь создадим ещё один граф, включающий все вершины нашего исходного набора, но добавим в него только те рёбра, которые являлись частью остовного дерева и весили меньше порогового уровня в 0.05 (выбран чисто эмпирически по гистограмме).

edges = [edge for edge in mst.edges\_iter(data=True) if edge[2]['weight'] <= limit]

 r.add\_edges\_from(edges)

 del s

В отличие от классического алгоритма MST, предпочтительнее отрезать рёбра не по количеству кластеров, а по пороговому уровню, как в иерархическом алгоритме. Это позволяет снизить влияние на результат кластеризации качество предварительной подготовки данных.

Осталось лишь отобразить результат. Для этого в NetworkX также есть встроенные средства, а именно функция draw\_graphviz. Интерес для нас представляет само остовное дерево и полученный граф кластеров.

pos = graphviz\_layout(g)

 nx.draw(g, pos, with\_labels=False, node\_size=3, prog='neato')(Рис.6)



Рисунок 5. Таблица весов ребер графа



Рисунок 6. Минимальное покрывающее дерево, и результат кластеризации

Данный метод выделил 5 кластеров входных данных, т.е упорядочил их по 5 группам.

В каждом кластере содержатся выделенные объекты.

**1-й кластер:**

1. **Москва**
2. **Санкт-Петербург**

**2-й кластер:**

1. **Московская область**
2. **Краснодарский край**

**3-й кластер:**

1. **Ростовская область**
2. **Башкортостан**
3. **Белгородская область**
4. **Татарстан**
5. **Челябинская**

**4-й кластер:**

1. **Омская область**
2. **Тюменская**
3. **Приморский край**
4. **Новосибирская**
5. **Кемеровская область**
6. **Алтайский край**
7. **Оренбургская область**
8. **Тульская область**
9. **Волгоградская область**
10. **Владимирская область**
11. **Крым**
12. **Свердловская область**
13. **Дагестан**
14. **Иркутская область**
15. **Пермский край**
16. **Саратовская область**
17. **Красноярский**
18. **Самарская область**
19. **Ставропольский**
20. **Воронежская область**
21. **Ленинградская область**

**5-й кластер:**

1. **Амурская область**
2. **Брянская область**
3. **Ямало-Немецкая АО ь**
4. **Севастополь**
5. **Мурманская**
6. **Костромская**
7. **Карачаево-Черкесская**
8. **Чеченская**
9. **Магаданская**
10. **Алтай**
11. **Еврейская АО**
12. **Чукотский АО**
13. **Ненецкий АО**
14. **Бурятия**
15. **Новгородская**
16. **Калининградская**
17. **Мордовия**
18. **Псковская**
19. **Якутия**
20. **Смоленская**
21. **Астраханская**
22. **Коми**
23. **Чувашия**
24. **Белгородская**
25. **Табовская**
26. **Волгоградская**
27. **Архангельская**
28. **Ивановская**
29. **Кировская**
30. **Калужская**
31. **Цльяновская**
32. **Курская**
33. **Ханты-Манстйская**
34. **Рязанская**
35. **Тверская**
36. **Пензская**
37. **Удмуртская**
38. **Хаабаровский край**
39. **Ярославская**
40. **Забайкальский край**
41. **Липецкая**
42. **Хакасия**
43. **Карелия**
44. **Тыва**
45. **Адыгея**
46. **Кабардино-Балканская**
47. **Сахалинская**
48. **Томская**
49. **Марий Эл**
50. **Орловская**
51. **Северная Осетия**
52. **Камчатский**
53. **Калмыкия**
54. **Ямало-Немецкая АО**
55. **Севстополь**
56. **Мурманская**
57. **Костромская**
58. **Карачаево-Черкесская**
59. **Чеченская**
60. **Магаданская**
61. **Алтай**
62. **Еврейская АО**
63. **Чукотский АО**
64. **Ненецкий АО**

# Глава 2. Прогнозирование

##  Прогнозирование ситуации

Прогнозирование временных рядов — это достаточно популярная аналитическая задача. Прогнозы используются, например, для понимания, сколько серверов понадобится online-сервису через год, каков будет спрос на каждый товар в гипермаркете, или для постановки целей и оценки работы команды (для этого можно построить baseline прогноз и сравнить фактическое значение с прогнозируемым).

Существует большое количество различных подходов для прогнозирования временных рядов, такие как [ARIMA](https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average), [ARCH](https://en.wikipedia.org/wiki/Autoregressive_conditional_heteroskedasticity), регрессионные модели, нейронные сети и т.д.

В нашей программе используется библиотека для прогнозирования временных рядов Facebook Prophet.

Библиотека fbprophet - это [additive regression model](https://en.wikipedia.org/wiki/Additive_model), состоящая из следующих компонент:

1. **Сезонные компоненты** s(t) отвечают за моделирование периодических изменений, связанных с недельной и годовой сезонностью. Недельная сезонность моделируется с помощью dummy variables. Добавляются 6 дополнительных признаков, например, [monday, tuesday, wednesday, thursday, friday, saturday], которые принимают значения 0 и 1 в зависимости от даты. Признак sunday, соответствующий седьмому дню недели, не добавляют, потому что он будут линейно зависеть от других дней недели и это будет влиять на модель.
Годовая же сезонность моделируется рядами Фурье.
2. **Тренд** g(t) — это кусочно-линейная или логистическая функция. С линейной функцией все понятно. Логистическая же функция вида

  позволяет моделировать рост с насыщением, когда при увеличении показателя снижается темп его роста. Типичный пример — это рост аудитории приложения или сайта.
Кроме всего прочего, библиотека умеет по историческим данным выбирать оптимальные точки изменения тренда. Но их также можно задать и вручную (например, если известны даты релизов новой функциональности, которые сильно повлияли на ключевые показатели).

1. Компонента h(t) отвечает за заданные пользователем **аномальные дни**, в том числе и нерегулярные, такие как, например, Black Fridays.
2. **Ошибка** ϵt содержит информацию, которая не учтена моделью.

В программе алгоритм реализован в модуле Predict.py.

Нам необходимо получить, и предобработать данные из входного файла Data1.csv.

df = pd.read\_csv('Data1.csv')

 df['y'] = np.log(df['y'])

 df.head()

Библиотека Prophet имеет интерфейс похожий на sklearn, сначала мы создаем модель, затем вызываем у нее метод fit и затем получаем прогноз. На вход методу fit библиотека принимает dataframe с двумя колонками:

* ds — время, поле должно быть типа date или datetime,
* y — числовой показатель, который мы хотим предсказывать.

Cоздаем объект класса Prophet (все параметры модели задаются в конструкторе класса, для начала возьмем default'ные параметры) и обучаем его.

m = Prophet()

 m.fit(df);

С помощью вспомогательной функции  Prophet.make\_future\_dataframe создаем dataframe, который содержит все исторические временные точки и еще 365 дней, для которых мы хотели построить прогноз.

Для того, чтобы построить прогноз вызываем у модели функцию predict и передаем в нее полученный на предыдущем шаге dataframe future.

future = m.make\_future\_dataframe(periods=365)

future.tail()

forecast = m.predict(future)

В библиотеке Prophet есть встроенные средства визуализации, которые позволяют оценить результат построенной модели.

Во-первых, метод Prophet.plot отображает прогноз. Честно говоря, в данном случае такая визуализация не очень показательна.

Вторая функция Prophet.plot\_components гораздо более полезная. Она позволяет посмотреть отдельно на компоненты: тренд, годовую и недельную сезонность. Если при построении модели были заданы аномальные дни/праздники, то они также будут отображаться на этом графике. (Рис.7)

forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail()

m.plot\_components(forecast)



Рисунок 6. График прогнозирования

##  Графическая составляющая приложения

Графическая часть реализована в главном файле Main.py при помощи библиотек PyQt5 и matplotlib. А также в файлах визуализации таблиц.

PyQt — набор «[привязок](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%B8%D0%B2%D1%8F%D0%B7%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)&action=edit&redlink=1)» [графического](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) [фреймворка](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) [Qt](https://ru.wikipedia.org/wiki/Qt) для [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [Python](https://ru.wikipedia.org/wiki/Python), выполненный в виде [расширения](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD) Python.

PyQt практически полностью реализует возможности Qt. А это более 600 классов, более 6000 функций и методов, включая:

1. существующий набор [виджетов](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B6%D0%B5%D1%82) графического интерфейса;
2. стили виджетов;
3. доступ к [базам данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) с помощью [SQL](https://ru.wikipedia.org/wiki/SQL) ([ODBC](https://ru.wikipedia.org/wiki/ODBC), [MySQL](https://ru.wikipedia.org/wiki/MySQL), [PostgreSQL](https://ru.wikipedia.org/wiki/PostgreSQL), [Oracle](https://ru.wikipedia.org/wiki/Oracle_%28%D0%A1%D0%A3%D0%91%D0%94%29));
4. QScintilla, основанный на [Scintilla](https://ru.wikipedia.org/wiki/Scintilla) виджет текстового редактора;
5. поддержку [интернационализации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) (i18n);
6. [парсер](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D1%81%D0%B5%D1%80) [XML](https://ru.wikipedia.org/wiki/XML);
7. поддержку [SVG](https://ru.wikipedia.org/wiki/SVG);
8. интеграцию с [WebKit](https://ru.wikipedia.org/wiki/WebKit), движком рендеринга HTML;
9. поддержку воспроизведения видео и аудио.

PyQt также включает в себя [Qt Designer](https://ru.wikipedia.org/wiki/Qt_Designer) ([Qt Creator](https://ru.wikipedia.org/wiki/Qt_Creator)) — дизайнер графического интерфейса пользователя. Программа pyuic генерирует Python код из файлов, созданных в Qt Designer. Это делает PyQt очень полезным инструментом для быстрого прототипирования. Кроме того, можно добавлять новые графические элементы управления, написанные на Python, в Qt Designer.

Matplotlib — библиотека на языке программирования [Python](https://ru.wikipedia.org/wiki/Python) для визуализации данных [двумерной (2D) графикой](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0#%D0%94%D0%B2%D1%83%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0_(2D)) ([3D графика](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0#%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) также поддерживается). Получаемые изображения могут быть использованы в качестве иллюстраций в публикациях.

Matplotlib является гибким, легко конфигурируемым пакетом, который вместе с [NumPy](https://ru.wikipedia.org/wiki/NumPy), [SciPy](https://ru.wikipedia.org/wiki/SciPy) и [IPython](https://ru.wikipedia.org/wiki/IPython) предоставляет возможности, подобные MATLAB. В настоящее время пакет работает с несколькими графическими библиотеками, включая [wxWindows](https://ru.wikipedia.org/wiki/WxWindows) и [PyGTK](https://ru.wikipedia.org/wiki/PyGTK).

Пакет поддерживает многие виды графиков и [диаграмм](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0):

* Графики (line plot)
* Диаграммы разброса (scatter plot)
* Столбчатые диаграммы (bar chart) и гистограммы (histogram)
* Круговые диаграммы (pie chart)
* Ствол-лист диаграммы (stem plot)
* Контурные графики (contour plot)
* Поля градиентов (quiver)
* Спектральные диаграммы (spectrogram)

В программе, наблюдается следующее их использование:

Происходит создание класса главного окна приложения, и его инициализация, и перерисовка его размеров.

class mainWindow(QtWidgets.QTabWidget):

 def \_\_init\_\_(self, parent = None):

 super(mainWindow, self).\_\_init\_\_(parent)

 self.setGeometry(0, 0, 800, 600)

Поскольку нам необходимо отображать наши графики, мы создаем график отображения, и поверхность для отрисовки графика:

self.fig = plt.figure()

self.canvas = FigureCanvas(self.fig)

Для вызова реализации методов кластеризации и прогнозирования, нам необходимо создать кнопки вызова данных методов, и связать их с вызовом соответствующих функций:

self.button = QtWidgets.QPushButton('Hierarchy', self)

self.button.clicked.connect(self.addHierarchy)

self.button1 = QtWidgets.QPushButton('KMeans', self)

self.button1.clicked.connect(self.addKMeans)

self.button2 = QtWidgets.QPushButton('MST', self)

self.button2.clicked.connect(self.addMst)

self.button3 = QtWidgets.QPushButton('Create prediction', self)

self.button3.clicked.connect(self.addPred)

А также выносим их на главный layout формы:

layout = QtWidgets.QVBoxLayout()

layout.addWidget(self.canvas)

layout.addWidget(self.button)

layout.addWidget(self.button1)

layout.addWidget(self.button2)

layout.addWidget(self.button3)

self.setLayout(layout)

Для вызова методов, и отображения таблиц, используются следующие вспомогательные функции:

 def addHierarchy(self):

 self.fig.clear()

 self.counts, self.bins, self.bars = hierarchy\_draw(names, data, plt, dist)

 self.canvas.draw()

 qm = QMessageBox

 ret = qm.question(self, '', "Are you want to show table?", qm.Yes | qm.No)

 if ret == qm.Yes:

 self.ShowHTable()

 def addKMeans(self):

 self.fig.clear()

 self.names, self.x, self.y, self.z = kmeans\_export(data, names, plt, self.fig)

 self.canvas.draw()

 qm = QMessageBox

 ret = qm.question(self, '', "Are you want to show table?", qm.Yes | qm.No)

 if ret == qm.Yes:

 self.ShowKTable()

 def addMst(self):

 self.fig.clear()

 self.counts, self.bins, self.bars = graph\_mst(dist, names, plt)

 self.canvas.draw()

 qm = QMessageBox

 ret = qm.question(self, '', "Are you want to show table?", qm.Yes | qm.No)

 if ret == qm.Yes:

 self.ShowHTable()

 def addPred(self):

 self.fig.clear()

 img = mpimg.imread('foo.png')

 imgplot = plt.imshow(img)

 fig\_size = [12, 9]

 plt.rcParams["figure.figsize"] = fig\_size

 self.canvas.draw()

 def ShowHTable(self):

 self.asd = HTable.Htbl(self.counts, self.bins, self.bars)

 self.asd.show()

 def ShowKTable(self):

 self.asd = KTable.Ktbl(self.names, self.x, self.y, self.z)

 self.asd.show()

##  Анализ модуля предсказаний

Наш прогноз получился вполне разумным, но имеет смысл сравнить его с классической моделью SARIMA - Seasonal autoregressive integrated moving average с таким же периодом.

Стоит отметить, что построение ARIMA модели требует гораздо больших затрат по сравнению с Prophet: нужно исследовать исходный ряд, привести его к стационарному, подобрать начальные приближения и потратить немало времени на подбор гипер-параметров алгоритма.

В случае алгоритма prophet мы получили качество около 37.35%. Но в данном случае усилия были не напрасны и предсказание SARIMA получилось более точным: MAPE=16.54%.

Но быстродействие Prophet позволяет списать на нет данную разбежку в точности, к тому же данную модель можно улучшить, без удара по производительности. Например, если предсказывать в этой библиотеке не исходный ряд, а после преобразования Бокса-Кокса, нормализующего дисперсию ряда, то мы получим прирост качества: MAPE=26.79%, что позволяет нам считать модель Prophet даже лучше чем ARIMA.



Рисунок 7. Визуализация Prophet модели



Рисунок 8.Визуализация всех моделей

##  Инструкция для пользователей (скриншоты)



Рисунок 9. Главное окно программы

При нажатии на кнопки, произойдут следующие действия:

* Hieratchу – иерархическая кластеризация
* KMeans – метод K-средних
* MST – алгоритм минимального покрывающего дерева
* Create prediction – построение прогноза

 

Рисунок 10.Иерархическая



Рисунок 11.К-средних



Рисунок 12. Минимальное остовное



Рисунок 13.Прогнозирование

#  Вывод

В результате работы программы, были получены статистические и графические результаты кластерного анализа данных, а также спрогнозированы события, на основе известных данных.На основе полученных данных иерархический метод и метод минимального остового дерева выделили 5 групп, а метод К-средних - семь групп. Следовательно, можно сделать вывод, что в условиях нашей задачи, первые два метода, оказались более точными.

Задача была решена полностью, при текущем уровне развития инструментов статистики и прогнозирования. Конечно, результат предсказания довольно далек от возможного результата, и возможно в будущем данная программа поможет в нахождении более совершенных методов построения модели предсказаний, однако в современных реалиях, задача выполнена максимально возможно, основываясь на следующем отношении: погрешность \* производительность.

 **Заключение**

В ходе выполнения работы были изучены основные математические и статистические библиотеки языка Python, графические библиотеки, ООП, среда разработки PyCharm, язык программирования Python.

Были получены следующие результаты:

 представление о принципе разработки готовых продуктов

 базис знаний, о кластерном анализе, его применении в цифровой экономике и построении моделей предсказаний

 полностью выполняющий условия технического задания продукт

Разработанное приложение позволяет строить прогноз, основываясь на входящих данных, а также выполнять кластерный анализ различными методами. В совокупности с удобным интерфейсом, и стабильной работоспособностью, оно способно стать полноценным торговым продуктом.

Поставленная в техническом задании цель полностью выполнена. Создано заявленное приложение. Также было выполнено необходимое количество тестов на следующих операционных системах: Windows 7, Windows 8, Windows 10. А также, на UNIX-подобных системах: Linux Mint, Linux Ubuntu, Arch Linux.

**Предложения**

 Огромная роль больших данных, цифровых технологий в трансформации социально-экономических систем очевидна, однако считаю актуальным мнение ряда авторов, что многие вопросы остаются еще слабо изученными в, частности, развитие цифрового потенциала с целью достижения инновационного роста отдельных фирм и отраслей, институциональные аспекты цифровой экономики, проблемы и перспективы развития бизнеса в условиях формирования цифровой экономики и др.

В связи с чем, считаю также актуальным дальнейшее совершенствование и применение разработанного программного продукта для анализа данных в различных областях жизнедеятельности и продолжение изучения и реализации проектов по этой перспективной для общества и интересной для меня теме. Надеюсь, что такая возможность мне представится, чему несомненно поспособствует дальнейшее целенаправленное обучение на направлении магистерской программы Цифровая экономика.

#  Список литературы

1. БЭРРИ, П. Изучаем программирование на Python

2. ВОРОНЦОВ, К. В. Лекции по алгоритмам кластеризации и многомерного шкалирования.

3. ВОРОНЦОВ, К. В. Машинное обучение. Курс лекций [Электронный ресурс]. http://www. machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf.

 4. Джулий Л.В., Емчук Л.В. Информационные системы и их роль в деятельности современных предприятий // В книге: Perspective economic and management issues Collection of scientific articles. Scientific journal «Economics and finance», «East West» Association For Advanced Studies and Higher Education. 2015. С. 130-134.

5**.** ЛИСИН А. В., ФАЙЗУЛЛИН Р. Т. Применение метаэвристических алгоритмов к решению задач кластеризации методом k-средних // Компьютерная оптика. — 2015. — Т. 39, №. 3. — С. 406–412.

6. НЕЙСКИЙ, И. М. Классификация и сравнение методов кластеризации // Интеллектуальные технологии и системы. Сборник учебно-методических работ и статей аспирантов и студентов. — М.: НОК «CLAIM», 2006. — Выпуск 8. — С. 130–142.

**7.** Попов Е.В., Семячков К.А., Симонова В.Л. Оценка влияния информационно-коммуникационных технологий на инновационную активность регионов // Финансы и кредит. 2016. № 46 (718). С. 46-60.

8. ТКАЧЕНКО, О. М. И ДР. Метод кластеризации на основе последовательного запуска k-средних с усовершенствованным выбором кандидата на новую позицию вставки // Научные труды Винницкого национального технического университета. — 2012. — №. 2.

9. ТХАНГ, В. В., ПАНТЮХИН, Д. В., ГАЛУШКИН, А. И. Гибридный алгоритм кластеризации FastDBSCAN // Труды Московского Физико-Технического Института. — 2015. — Т. 7, №. 3. — С. 77–81.

10. AMORIM DE, R. C., HENNIG, C. Recovering the number of clusters in data sets with noise features using feature rescaling factors // Information Sciences. — 2015. — Vol. 324. — P. 126– 145.

11. ARTHUR, D., VASSILVITSKII, S. k-means++: The advantages of careful seeding // Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. — Society for Industrial and Applied Mathematics, 2007. — P. 1027–1035.

12. CHARIKAR, M. ET AL. Incremental clustering and dynamic information retrieval // SIAM Journal on Computing. — 2004. — Vol. 33, №. 6. — P. 1417–1440.

13**.** COMANICIU, D., MEER, P. Mean shift: A robust approach toward feature space analysis // Pattern Analysis and Machine Intelligence, IEEE Transactions on. — 2002. — Vol. 24, №. 5. — P. 603–619.

 14. DUMAN, S., GUVEN ¸C, U., Y ¨ OR¨ UKEREN, N. ¨ Gravitational Search Algorithm for Economic Dispatch with Valve-Point Effects // International Review of Electrical Engineering 2010. — 2010. — Vol. 5. — P. 2890–2895.

15**.** ELKAN, C. Using the triangle inequality to accelerate kmeans // ICML. — 2003. — Vol. 3. — P. 147–153.

16. ESTIVILL-CASTRO, V., LEE, I. Argument free clustering for large spatial point-data sets via boundary extraction from 26 Delaunay Diagram // Computers, Environment and urban systems. — 2002. — Vol. 26, №. 4. — P. 315–334.

17. ESTIVILL-CASTRO, V., LEE, I. Clustering with obstacles for geographical data mining // ISPRS Journal of Photogrammetry and Remote Sensing. — 2004. — Vol. 59, №. 1. — P. 21–34.

18. FRALEY, C., RAFTERY, A. E. Model-based clustering, discriminant analysis, and density estimation // Journal of the American statistical Association. — 2002. — Vol. 97, №. 458. — P. 611–631.

19. GeographicLib — a small set of C++ classes for converting between geographic, UTM, UPS, MGRS, and geocentric coordinates [Электронный ресурс]. — Режим доступа: http://geographiclib.sourceforge.net/ (дата обращения: 14.11.2015).

20. GOLUBEV, A., CHECHETKIN, I., SOLNUSHKIN, K.S., SADOVNIKOVA, N., PARYGIN, D., SHCHERBAKOV, M. Strategway: web solutions for building public transportation routes using big geodata analysis // Proceedings of the 17th International Conference on Information Integration and Webbased Applications & Services. — ACM, 2015. — P. 91–94.

21. HAN, J., KAMBER, M., TUNG, A. K. H. Spatial Clustering Methods in Data Mining: A Survey // Geographic Data Mining and Knowledge Discovery, Research Monographs in GIS. — 2001. — P. 201–231.

22. HUANG, Z. A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining // DMKD. — 1997. — 8 p.

23. HUSSEIN, N. A Fast Greedy k-means Algorithm: Master’s Thesis. // University of Amsterdam, Amsterdam. — 2002.

 24. KANUNGO, T. ET AL. An efficient k-means clustering algorithm: Analysis and implementation // Pattern Analysis and Machine Intelligence, IEEE Transactions on. — 2002. — Vol. 24, №. 7. — P. 881–892.

25. KARNEY, C. F. F. Algorithms for geodesics // Journal of 27 Geodesy. — 2013. — Vol. 87, №. 1. — P. 43–55.

26. KOPERSKI, K., HAN, J., ADHIKARY, J. Mining knowledge in geographical data // Communications of ACM. — 1998. — Vol. 26.

27. KRISHNA, K., MURTY, M. N. Genetic k-means algorithm //Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on. — 1999. — Vol. 29, №. 3. — P. 433–439.

28. LAI, J. Z. C., HUANG, T. J., LIAW, Y. C. A fast k-means clustering algorithm using cluster center displacement // Pattern Recognition. — 2009. — Vol. 42, №. 11. — P. 2551– 2556.

29. LIKAS, A., VLASSIS, N., VERBEEK, J. J. The global kmeans clustering algorithm // Pattern recognition. — 2003. — Vol. 36, №. 2. — P. 451–461.

30. LLET´I, R. ET AL. Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes // Analytica Chimica Acta. — 2004. — Vol. 515, №. 1. — P. 87–100.

31. NISTER, D., STEWENIUS, H. Scalable recognition with a vocabulary tree // Computer vision and pattern recognition, 2006 IEEE computer society conference on.

32. S. Aleksandrovich., N. Advancing the Digital Economy into the 21st Century //2017 New York: Bantam. Lane //Information Systems Frontiers 1:3, 317-320 (1999);

33. Sean J. Taylor, Benjamin Letham ["Forecasting at scale"](https://facebookincubator.github.io/prophet/static/prophet_paper_20170113.pdf).

# Приложение

**Main.py**

#! /usr/bin/env python

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

import matplotlib.pyplot as plt

from PyQt5.QtWidgets import QMessageBox

from scipy.spatial.distance import pdist

from hierarchyCls import hierarchy\_draw

from KMeansCls import kmeans\_export

from MstCls import graph\_mst

from shared import get\_data

#from Predict import Pred

import HierarchyTable as HTable

import KMeansTable as KTable

import sys

from PyQt5 import QtCore, QtGui, QtWidgets, uic

from matplotlib.backends.backend\_qt4agg import FigureCanvasQTAgg as FigureCanvas

import matplotlib.image as mpimg

class mainWindow(QtWidgets.QTabWidget):

 def \_\_init\_\_(self, parent = None):

 super(mainWindow, self).\_\_init\_\_(parent)

 self.setGeometry(0, 0, 800, 600)

 self.names = []

 self.x = []

 self.y = []

 self.z = []

 self.fig = plt.figure()

 self.canvas = FigureCanvas(self.fig)

 self.button = QtWidgets.QPushButton('Hierarchy', self)

 self.button.clicked.connect(self.addHierarchy)

 self.button1 = QtWidgets.QPushButton('KMeans', self)

 self.button1.clicked.connect(self.addKMeans)

 self.button2 = QtWidgets.QPushButton('MST', self)

 self.button2.clicked.connect(self.addMst)

 self.button3 = QtWidgets.QPushButton('Create prediction', self)

 self.button3.clicked.connect(self.addPred)

 layout = QtWidgets.QVBoxLayout()

 layout.addWidget(self.canvas)

 layout.addWidget(self.button)

 layout.addWidget(self.button1)

 layout.addWidget(self.button2)

 layout.addWidget(self.button3)

 self.setLayout(layout)

 def addHierarchy(self):

 self.fig.clear()

 self.counts, self.bins, self.bars = hierarchy\_draw(names, data, plt, dist)

 self.canvas.draw()

 qm = QMessageBox

 ret = qm.question(self, '', "Are you want to show table?", qm.Yes | qm.No)

 if ret == qm.Yes:

 self.ShowHTable()

 def addKMeans(self):

 self.fig.clear()

 self.names, self.x, self.y, self.z = kmeans\_export(data, names, plt, self.fig)

 self.canvas.draw()

 qm = QMessageBox

 ret = qm.question(self, '', "Are you want to show table?", qm.Yes | qm.No)

 if ret == qm.Yes:

 self.ShowKTable()

 def addMst(self):

 self.fig.clear()

 self.counts, self.bins, self.bars = graph\_mst(dist, names, plt)

 self.canvas.draw()

 qm = QMessageBox

 ret = qm.question(self, '', "Are you want to show table?", qm.Yes | qm.No)

 if ret == qm.Yes:

 self.ShowHTable()

 def addPred(self):

 self.fig.clear()

 img = mpimg.imread('foo.png')

 imgplot = plt.imshow(img)

 fig\_size = [12, 9]

 plt.rcParams["figure.figsize"] = fig\_size

 self.canvas.draw()

 def ShowHTable(self):

 self.asd = HTable.Htbl(self.counts, self.bins, self.bars)

 self.asd.show()

 def ShowKTable(self):

 self.asd = KTable.Ktbl(self.names, self.x, self.y, self.z)

 self.asd.show()

def main():

 app = QtWidgets.QApplication(sys.argv)

 main = mainWindow()

 main.show()

 sys.exit(app.exec\_())

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

 names, data = get\_data()

 dist = pdist(data, 'euclidean')

 main()

**hierarchyCls.py**

#! /usr/bin/env python

from scipy.cluster import hierarchy

def hierarchy\_draw(labels, data, plt, dist):

 plt.subplot(2, 1, 1)

 counts, bins, bars = plt.hist(dist, 500, color='green', alpha=0.5)

 Z = hierarchy.linkage(dist, method='average')

 level = .25

 plt.subplot(2, 1, 2)

 hierarchy.dendrogram(Z, labels=labels, color\_threshold=level, leaf\_font\_size=5, count\_sort=True)

 return counts, bins, bars

**hierarchyTable.py**

#! /usr/bin/env python

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

from PyQt5.QtCore import QSize, Qt

from PyQt5.QtWidgets import QDialog, QWidget, QGridLayout, QTableWidget, QMainWindow, QTableWidgetItem

class Htbl(QMainWindow):

 def \_\_init\_\_(self, counts, bins, bars, parent = None):

 super(Htbl, self).\_\_init\_\_(parent)

 self.setMinimumSize(QSize(640 , 80))

 self.setWindowTitle("Hierarchy/ Mst")

 central\_widget = QWidget(self)

 self.setCentralWidget(central\_widget)

 grid\_layout = QGridLayout()

 central\_widget.setLayout(grid\_layout)

 table = QTableWidget(self)

 table.setColumnCount(3)

 table.setRowCount(len(counts))

 table.setHorizontalHeaderLabels(["Counts", "Bins", "Bars"])

 table.horizontalHeaderItem(0).setTextAlignment(Qt.AlignHCenter)

 table.horizontalHeaderItem(1).setTextAlignment(Qt.AlignHCenter)

 table.horizontalHeaderItem(2).setTextAlignment(Qt.AlignHCenter)

 i = 0

 for value in counts:

 table.setItem(i, 0, QTableWidgetItem(str(value)))

 i = i + 1

 i = 0

 for value in bins:

 table.setItem(i, 1, QTableWidgetItem(str(value)))

 i = i + 1

 i = 0

 for value in bars:

 table.setItem(i, 2, QTableWidgetItem(str(value)))

 i = i + 1

 table.resizeColumnsToContents()

 grid\_layout.addWidget(table, 0, 0)

**KMeansCls.py**

#! /usr/bin/env python

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

from collections import deque

import numpy

from scipy.cluster import \*

from scipy.spatial.distance import cdist

from mpl\_toolkits.mplot3d import Axes3D

names = []

X = []

Y = []

Z = []

def kmeans\_export(data, labels, plt, fig):

 centroids = vq.kmeans(numpy.array(data), 7, iter=200)[0]

 res = [[] for i in xrange(len(centroids))]

 d = cdist(numpy.array(data), centroids, 'euclidean')

 for i, l in enumerate(d):

 res[l.tolist().index(l.min())].append((labels[i], data[i]))

 kmeans\_draw(res, plt, fig)

 return names, X, Y, Z

def kmeans\_draw(clusters, plt, fig):

 colors = deque(['r', 'g', 'b', 'c', 'm', 'y', 'k'])

 ax = Axes3D(fig)

 for cluster in clusters:

 color = colors.popleft()

 for name, coord in cluster:

 x, y, z = coord

 names.append(name)

 X.append(x)

 Y.append(y)

 Z.append(z)

 ax.plot3D([x], [y], [z], marker='o', c=color)

 ax.set\_xlabel(u'ДТП')

 ax.set\_ylabel(u'Погибло')

 ax.set\_zlabel(u'Ранено')

**KMeansTable.py**

#! /usr/bin/env python

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

from PyQt5.QtCore import QSize, Qt

from PyQt5.QtWidgets import QDialog, QWidget, QGridLayout, QTableWidget, QMainWindow, QTableWidgetItem

class Ktbl(QMainWindow):

 def \_\_init\_\_(self, names, x, y, z, parent = None):

 super(Ktbl, self).\_\_init\_\_(parent)

 self.setMinimumSize(QSize(640 , 80))

 self.setWindowTitle("KMeans")

 central\_widget = QWidget(self)

 self.setCentralWidget(central\_widget)

 grid\_layout = QGridLayout()

 central\_widget.setLayout(grid\_layout)

 table = QTableWidget(self)

 table.setColumnCount(4)

 table.setRowCount(len(names))

 table.setHorizontalHeaderLabels(["Name", "X", "Y", "Z"])

 table.horizontalHeaderItem(0).setTextAlignment(Qt.AlignHCenter)

 table.horizontalHeaderItem(1).setTextAlignment(Qt.AlignHCenter)

 table.horizontalHeaderItem(2).setTextAlignment(Qt.AlignHCenter)

 i = 0

 for value in names:

 table.setItem(i, 0, QTableWidgetItem(value))

 i = i + 1

 i = 0

 for value in x:

 table.setItem(i, 1, QTableWidgetItem(str(value)))

 i = i + 1

 i = 0

 for value in y:

 table.setItem(i, 2, QTableWidgetItem(str(value)))

 i = i + 1

 i = 0

 for value in z:

 table.setItem(i, 3, QTableWidgetItem(str(value)))

 i = i + 1

 table.resizeColumnsToContents()

 grid\_layout.addWidget(table, 0, 0)

**MstCls.py**

#! /usr/bin/env python

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

from collections import deque

import networkx as nx

from networkx.drawing.nx\_agraph import graphviz\_layout

def graph\_draw(g, plt):

 pos = graphviz\_layout(g)

 nx.draw(g, pos, with\_labels=False, node\_size=3, prog='neato')

def graph\_mst(dist, labels, plt):

 limit = 0.05

 s = nx.Graph() # исходный граф

 s.add\_nodes\_from(labels)

 r = s.copy() # результат кластеризации

 dq = deque(dist)

 len\_x = len(labels)

 for x in xrange(len\_x - 1):

 for y in xrange(x + 1, len\_x):

 s.add\_edge(labels[x], labels[y], weight=dq.popleft())

 mst = nx.minimum\_spanning\_tree(s)

 counts, bins, bars = plt.hist([edge[2]['weight'] for edge in mst.edges\_iter(data=True)], 100, color='red', alpha=0.3)

 edges = [edge for edge in mst.edges\_iter(data=True) if edge[2]['weight'] <= limit]

 r.add\_edges\_from(edges)

 del s

 plt.subplot(2, 1, 1)

 plt.title(u"Минимальное остовное дерево графа")

 graph\_draw(mst, plt)

 plt.subplot(2, 1, 2)

 plt.title(u"Результат кластеризации")

 graph\_draw(r, plt)

 return counts, bins, bars

**Predict.py**

#! /usr/bin/env python

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

import pandas as pd

import numpy as np

from fbprophet import Prophet

def Pred(plt):

 df = pd.read\_csv('Data1.csv')

 df['y'] = np.log(df['y'])

 df.head()

 m = Prophet()

 m.fit(df);

 future = m.make\_future\_dataframe(periods=365)

 future.tail()

 forecast = m.predict(future)

 forecast[['ds', 'yhat', 'yhat\_lower', 'yhat\_upper']].tail()

 m.plot\_components(forecast)

**shared.py**

#! /usr/bin/env python

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

\_\_author\_\_ = 'esemi'

import numpy as np

def get\_data():

 source = [row.strip().split(';') for row in open('Data.csv')]

 names = [row[0] for row in source[1:]]

 data = [map(float, row[1:]) for row in source[1:]]

 return names, norm(data)

def norm(data):

 matrix = np.array(data, "f")

 len\_val = len(matrix[1, :])

 for i in range(len\_val):

 local\_min = matrix[:, i].min()

 if local\_min != 0.0:

 matrix[:, i] -= local\_min

 local\_max = matrix[:, i].max()

 if local\_max != 0.0:

 matrix[:, i] /= local\_max

 return matrix.tolist()

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

 get\_data()