

Министерство образования и науки Российской Федерации

**Южно-Российский государственный политехнический
университет (НПИ) имени М.И. Платова**

Д.В. Шайхутдинов, Н.Д. Наракидзе

**ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ И
ТЕХНОЛОГИИ**

Учебно-методическое пособие к практическим занятиям

**Новочеркасск
ЮРГПУ(НПИ)
2016**

УДК 389(075.8)

Рецензент – канд. техн. наук, доц. В.А. Кучеров

Шайхутдинов Д.В., Наракидзе Н.Д.

Интеллектуальные информационные и измерительные системы: учебно-методическое пособие к лабораторным занятиям / Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова – Новочеркасск: ЮРГПУ(НПИ) 2016. –

Пособие содержит четыре работы, посвященные соответственно: передаче информации между средствами измерений и блоками сбора данных, обработке данных алгоритмами машинного обучения и отображения данных. К каждой работе приведена программа выполнения и пояснения к работе, контрольные вопросы и перечень необходимой литературы.

Пособие предназначено для студентов, обучающихся по направлению подготовки «Приборостроение». Может быть полезно студентам родственных направлений.

Пособие подготовлено в рамках работ, проводимых ЦКП «Диагностика и энергоэффективное электрооборудование». Пособие подготовлено при финансовой поддержке РФФИ в рамках научного проекта № 18-38-20188.

УДК 389(075.8)

© Южно-Российский государственный
политехнический университет (НПИ)
имени М.И. Платова, 2016

ОБЩИЕ ТРЕБОВАНИЯ

Подготовка к работе проводится заранее и состоит из изучения описания практической работы, рекомендованной литературы, составления таблиц.

Все работающие в аудитории студенты обязаны ознакомиться с правилами техники безопасности и после инструктажа преподавателя расписаться в журнале по технике безопасности. Включение компьютеров допускается только с разрешения преподавателя.

Отчет по практической работе составляется каждым студентом индивидуально и должен содержать: наименование, дату выполнения, программу работы, необходимые таблицы, типы используемого оборудования, результаты исследования и расчеты. Отчет оформляется в соответствии с требованиями, указанными в [1, 2]. В отчете должны быть приведены выводы о проделанной работе.

Практическая работа №1

Работа с последовательным портом средствами Python

Цель работы: получить навыки использования последовательного порта передачи данных в режиме чтения средствами языка программирования Python и реализовать соответствующее приложение в виде независимого от ОС контейнера Docker, размещенного в виде open source проекта в репозитории Github.

Работа разделена на три части:

- Работа с Git и Github.
- Основы работы с технологиями контейнеризации Docker.
- Создание программы для работы с последовательным портом средствами Python.

Часть 1

Основы Git и Github

Выполнение данной части практической работы направлено на изучение:

1. наиболее распространенных практик в области контроля версий программного обеспечения, его использования в командной разработке ПО и DevOps;
2. концепции Git, основанной на понятиях репозитория и ветвления версий ПО;
3. порядка использования GitHub и его базовых операций.

Основные идеи и теоретические основы.

Разработчики приложений редко работают в одиночку. В крупных проектах в области веб-/облачных/мобильных приложений, в проектах искусственного интеллекта, обычно задействовано множество людей – разработчики интерфейса, серверной части, баз данных и многие другие. Вносимые каждым участником изменения обязательно должны отслеживаться и контролироваться руководителем, чтобы исключить значимые ошибки в условиях совместной работы. Для организации такой работы применяются технологии распределенного контроля версий.

История. Разработка Linux в начале 2000-х годов осуществлялась в рамках бесплатной системы, известной как BitKeeper. В 2005 году BitKeeper превратился в платную систему, что создало определенные проблемы для разработчиков Linux. Линус Торвальдс возглавил команду по разработке новой системы контроля версий исходного кода. Проект реализовывался в короткие сроки, а его ключевые характеристики были определены в следующем виде:

- поддержка нелинейного развития (в то время патчи для Linux приходили со скоростью 6,7 патчей в секунду, необходимо было обеспечить эффективное решение связанных с данной особенностью задач);
- распределенная разработка: каждый разработчик должен иметь локальную копию полной истории разработки;
- совместимость с существующими системами и протоколами, что необходимо для обеспечения разнообразия сообщества Linux;
- эффективное управление крупными проектами;
- аутентификация истории, что гарантирует, что все распределенные системы будут иметь идентичные обновления кода;
- подключаемые стратегии слияния: множество путей развития могут привести к сложным интеграционным решениям, которые могут потребовать четких интеграционных стратегий.

Распределенные системы контроля версий (Distributed Version Control Systems, DVCS) стали важнейшими инструментами разработки программного обеспечения для совместной разработки ПО. Их используют не только инженеры-программисты и специалисты DevOps, но и многие другие

специалисты, в том числе, в области Data Science. Они полезны везде, где требуется отслеживание изменений/версий и/или сотрудничество между несколькими пользователями.

Несмотря на то, что существует множество распределенных систем управления версиями, Git является одной из самых популярных, а GitHub – наиболее популярной платформой контроля версий на базе Git. Популярность Git и GitHub делает их важным навыком для специалистов, связанных с разработкой ПО.

В данной практической работе ставятся следующие задачи:

1. развитие концептуальных и практических навыков для работы с Git и GitHub, начиная с обзора Git и GitHub, создания учетной записи GitHub, репозитория проекта, добавления в него файлов, работы с помощью веб-интерфейса;
2. знакомство с базовыми процессами Git, включающими branche, pull requests и merge, операции fork и clone для общедоступных репозиториев, pull и push для синхронизации базы кода между локальными и удаленными репозиториями;
3. практика работы с командами Git для процесса совместной разработки.

Система контроля версий позволяет отслеживать изменения в ваших документах. Это позволяет восстановить старые версии документа, если была допущена ошибка и значительно упрощает сотрудничество с другими разработчиками. Git – бесплатное программное обеспечение с открытым исходным кодом, распространяемое по лицензии GNU General Public License. Git – это распределенная система контроля версий, что означает, что пользователи в любой точке мира могут получить копию вашего проекта на своем компьютере. После того, как они внесли изменения, они могут синхронизировать свою версию с удаленным сервером, чтобы поделиться ею с вами. Git – не единственная существующая система контроля версий, но данная технология фактически единственная, которая обеспечивает аспект распределенной работы, что и стало основной причиной ее значительной популярности. Системы контроля версий широко используются для задач, связанных с кодом, но вы также можете управлять изображениями, документами и любыми другими типами файлов. Вы можете использовать Git без веб-интерфейса, используя интерфейс командной строки.

GitHub – один из самых популярных веб-сервисов для репозиториев Git. К популярным также необходимо отнести GitLab, BitBucket и Beanstalk. Что особенного в модели Git Repository?

- Git спроектирован как распределенная система контроля версий.
- Git в первую очередь сосредоточен на отслеживании исходного кода во время разработки.
- Git содержит элементы для координации действий программистов, отслеживания изменений и поддержки нелинейных рабочих процессов.

Git – это распределенная система контроля версий, которая используется для отслеживания изменений контента. Она служит центральной точкой для сотрудничества. В централизованной системе контроля версий каждый разработчик должен извлечь код из центральной системы и снова внести в него изменения. Поскольку Git представляет собой распределенную систему контроля версий, у каждого разработчика есть локальная копия полной истории разработки, и изменения копируются из одного такого репозитория в другой. Каждый разработчик может выступать в роли хаба (центрального узла).

При правильном использовании Git существует основная ветка (branch), соответствующая коду, запускаемому по умолчанию. Команды могут постоянно интегрировать изменения, готовые к выпуску, и одновременно работать над отдельными ветвями между выпусками в свет основной. Git также позволяет централизованно администрировать задачи с контролем уровня доступа для каждой команды. Git может существовать локально, например, через клиент GitHub Desktop, или его можно использовать напрямую через браузер, подключенный к веб-интерфейсу GitHub.

GitHub – это онлайн-хостинг для репозиториев Git. GitHub – продукт дочерней компании Microsoft. GitHub предлагает бесплатные, профессиональные и корпоративные учетные записи. По состоянию на август 2019 года у GitHub было более 100 миллионов репозиториев.

Репозиторий – это структура данных для хранения документов, включая исходный код приложения. Репозиторий может отслеживать и поддерживать контроль версий.

Стоит остановиться также на GitLab. GitLab – это полноценная платформа DevOps, представленная в виде одного приложения. GitLab предоставляет доступ к репозиториям Git, контролируемым системой управления исходным кодом. С GitLab разработчики могут:

- Просматривать код, оставляя комментарии и помогая улучшать код друг друга.
- Работать со своей локальной копией кода.
- Осуществлять ветвления и слияния при необходимости.
- Оптимизировать тестирование и доставку с помощью встроенной непрерывной интеграции (CI) и непрерывной доставки (CD).

Прежде чем приступить к работе, вам необходимо знать несколько основных терминов:

Протокол SSH – метод безопасного удаленного входа с одного компьютера на другой.

Репозиторий в простом варианте изложения является местом, которое содержит папки вашего проекта, для которых настроен контроль версий.

Форк / fork – копия репозитория.

pull request – способ запроса на то, чтобы кто-то рассмотрел и утвердил предложенные вами изменения.

working directory содержит файлы и подкаталоги на вашем компьютере, которые связаны с Git репозиторием.

Есть несколько основных команд Git, которые вы будете использовать. Когда вы начинаете работу с новым репозиторием, его создание происходит только один раз: либо локально с последующей отправкой на GitHub, либо путем клонирования существующего репозитория с помощью команды «`git init`».

`git add` перемещает изменения из рабочего каталога в промежуточную область. `git status` позволяет вам видеть состояние вашего рабочего каталога и промежуточный snapshot ваших изменений.

`git commit` делает снимок / snapshot изменений и фиксирует их в проекте.

`git reset` отменяет изменения, внесенные вами в файлы в вашем рабочем каталоге.

`git log` позволяет просматривать предыдущие изменения в проекте.

`git branch` позволяет вам создать изолированную среду в вашем репозитории для внесения изменений.

`git checkout` позволяет просматривать и изменять существующие изолированные среды (*branch'и*).

`git merge` позволяет соединить *branch'и* воедино.

Все файлы в GitHub хранятся в отдельных *branch*. Разработчики работают над исходными файлами в *branch*. *Master branch* является основной. Она хранит развертываемую версию вашего кода. *Master branch* создается по умолчанию, однако, вы можете использовать любую ветку в качестве основной, готовой к запуске кода. Когда вы планируете что-то изменить, вы создаете новую *branch* и даете ей понятное имя. Новая *branch* создается как точная копия исходной *branch*. Когда вы вносите изменения, созданная вами *branch* сохраняет изменения.

Чтобы изменить содержимое файла в теории Git необходимо не просто внести изменения, но и выполнить `commit` изменений. Когда разработчик завершает порученную ему работу, для сохранения изменений он выполняется т.н. `commit` / фиксацию кода. `Commit` означает, что разработчик убежден, что код представляет собой стабильную основу для разрабатываемой функции или набора функций. Когда разработчик фиксирует измененный исходный код, он должен написать комментарий, описывающий изменения. Комментарий должен быть содержательным. Некоторые дополнительные «правила приличия»:

- Не заканчивайте сообщение при коммите точкой.
 - Сообщения о коммитах должны содержать не более 50 символов – для предоставления подробной информации используйте расширенное окно.
- Разработчики могут изменять исходные файлы в ветке, но изменения не будут опубликованы до утверждения. Порядок работы с утверждением:

- Подается команда pull.
- Код рассматривается и утверждается.
- Утвержденный код объединяется с основным кодом.

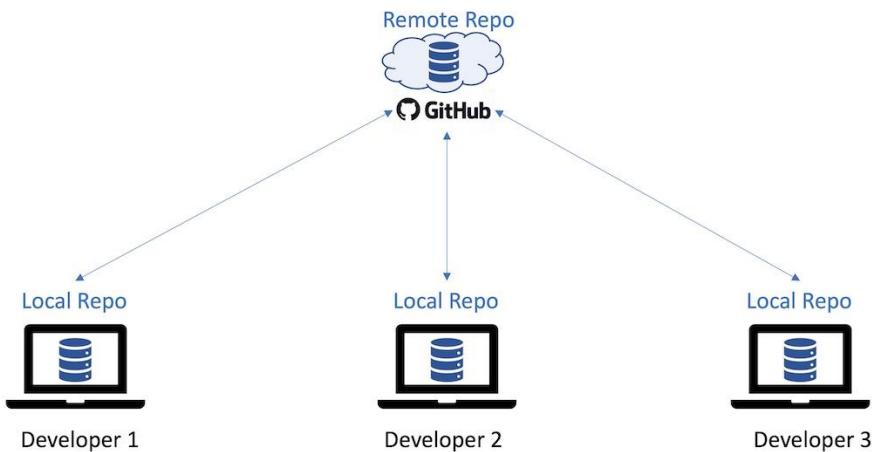
Когда все изменения в ветке завершены, эта ветка считается устаревшей и ее следует удалить.

Pull используется для инициирования слияния ветвей с целью фиксации изменений. *pull request* делает предложенные изменения доступными для просмотра и использования другими. Pull может следовать за любыми коммитами, даже если код еще не завершен. Для pull требуется, чтобы специальный пользователь утвердил изменения. Это может быть автор изменения или он может быть назначен из состава команды. Обратите внимание, что GitHub автоматически отправляет pull request от вашего имени, если вы вносите изменения в branch, которой вы не владеете. Поскольку файлы журналов неизменяемы, всегда можно найти человека, утвердившего внесение изменений. Чтобы открыть новый pull request в веб-интерфейсе GitHub необходимо выполнить следующую последовательность действий:

- Нажмите «Pull request» и выберите «New pull request».
- Выберите новую branch в поле сравнения.
- Прокрутите вниз, чтобы просмотреть изменения.
- Подтвердите, что изменения – это то, что вы хотите предоставить к утверждению.
- Добавьте заголовок и описание к запросу.
- Нажмите «Create pull request».

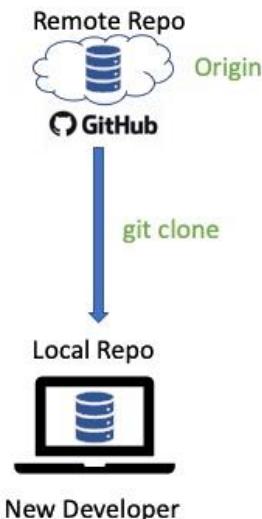
Forking и Cloning GitHub репозиториев.

Системы контроля версий с распределенным исходным кодом, такие как GitHub, позволяют нескольким разработчикам проектов параллельно работать над его кодовой базой. Проект может находиться на GitHub.com в качестве общедоступного или частного репозитория. Каждый разработчик, работающий над проектом, может иметь на своих компьютерах собственные копии репозитория. Копия репозитория на компьютере разработчика является локальной для этого разработчика, и, следовательно, этот разработчик также называет это репозиторий своим *local repo*. Копия репозитория на GitHub.com находится на удаленном сервере, и, следовательно, для каждого разработчика это *remote repo*.

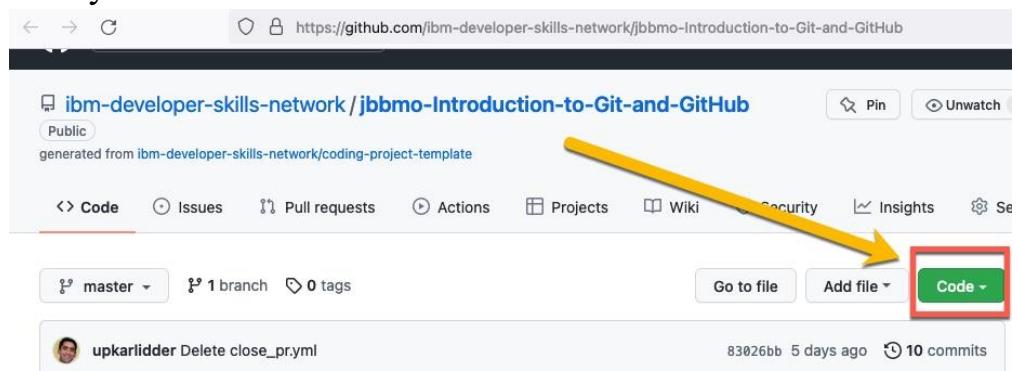


Clone

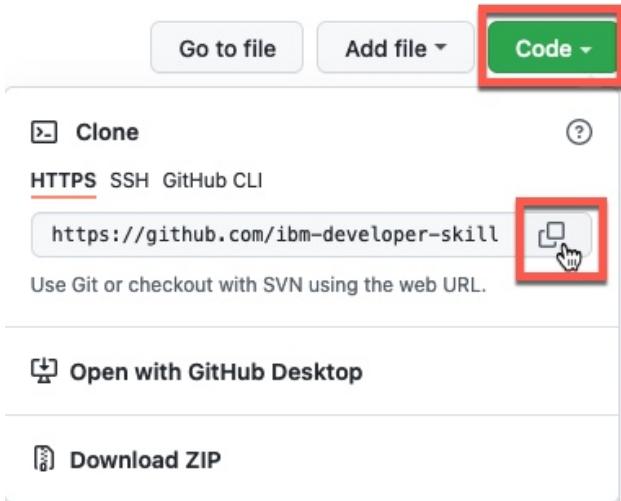
Допустим, к команде присоединяется новый разработчик для совместной работы над проектом. Этот разработчик может создать идентичную копию remote repo, используя операцию git clone. Удаленный репозиторий, из которого изначально клонирован проект, также называется origin.



Любой репозиторий в GitHub можно клонировать, перейдя в репозиторий и нажав кнопку «Code».



После этого у вас будет возможность получить всю кодовую базу удаленного репозитория несколькими способами, включая возможность скопировать URL-адрес HTTPS, а затем указать скопированный URL-адрес для выполнения команды git clone URL с вашего локального компьютера.



Создание ветвей и синхронизация изменений

После клонирования репозитория на локальный компьютер разработчик может начать вносить изменения в базу кода. Это может быть для таких задач, как добавление функций и улучшений или исправление ошибок. Обычно разработчик использует `git branch` для создания ветки, например, `feature1-branch`, делает эту ветку активной с помощью `git checkout` и вносит изменения в эту ветку, например, добавив или отредактировав файлы. Разработчик часто сохраняет свои изменения внутри ветки, используя `git add`, за которым следует `git commit`.

После того, как изменения для конкретной ветки завершены, вместо непосредственного слияния с основной веткой часто рекомендуется выполнить `push` новой ветки с изменениями в `origin`, где другие разработчики/рецензенты смогут протестировать/проверить изменения перед слиянием ветки с основной.

ПРИМЕЧАНИЕ. Поскольку в этом сценарии `feature1-branch` была разработана новым разработчиком проекта, у этого разработчика может не быть доступа для объединения своей ветки с основной веткой. Фактически, во многих проектах только сопровождающим или администраторам проекта разрешено объединяться с основной веткой, а в некоторых случаях может потребоваться экспертная оценка. Чтобы запросить проверку и объединение ваших изменений с основной веткой, многие проекты требуют отправки Pull Request (PR). Тогда как в некоторых случаях, например, если вы единственный разработчик проекта, этот шаг PR можно пропустить, и вы можете объединить и отправить свои изменения напрямую, если у вас есть доступ на запись к исходному репозиторию.

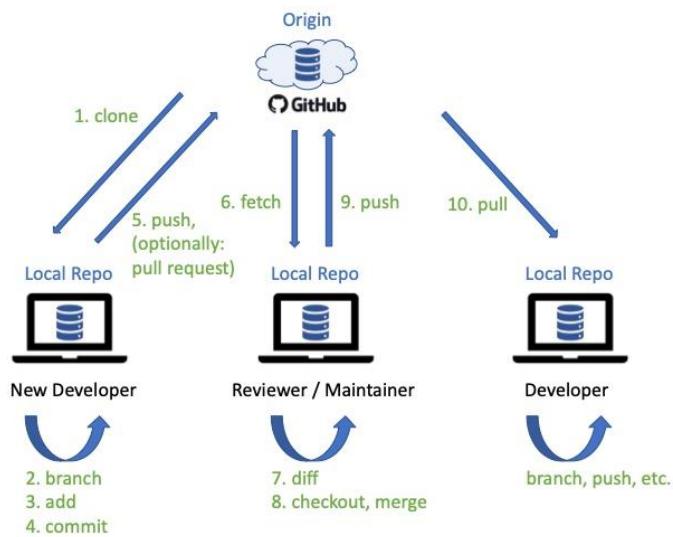
Время от времени разработчик может захотеть получить последнюю копию репозитория из `origin`, чтобы использовать ее в качестве основы для внесения изменений или проверки изменений другими. Например, это может произойти после того, как изменения в `feature1-branch` были отправлены в источник, и самостоятельный разработчик хочет просмотреть код. Для этой цели можно использовать команду `git fetch`.

Команда `git diff` может помочь другим разработчикам, просматривающим ваш код, выявить и сравнить изменения. Как только рецензент или сопровождающий проект просмотрит изменения и будет удовлетворен, он выполнит `git checkout` – перейдет в основную ветку, а затем `git merge` новую `feature1-branch`, которую затем можно будет удалить. После локального слияния ветки рецензент может с помощью `git push` обновленную основную ветку обратно в `origin`.

ПРИМЕЧАНИЕ. Команду «`git-remote -v`» можно использовать для проверки того, с какими удаленными репозиториями вы синхронизируете изменения `push` и `fetch`.

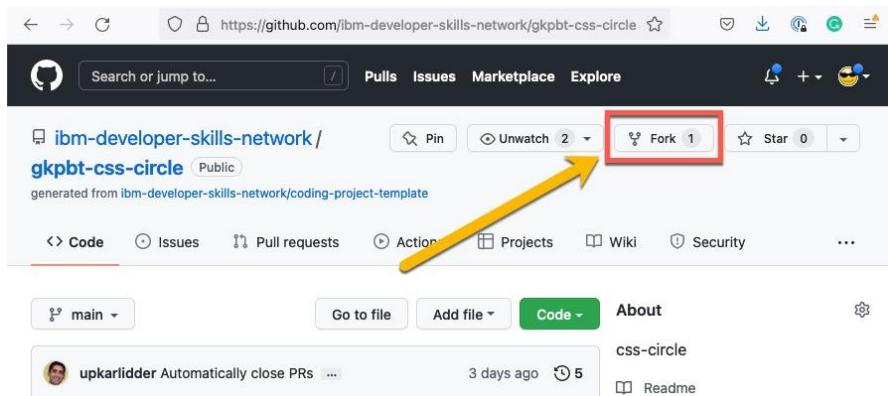
Другой вариант получения последней копии репозитория – использовать команду `git pull`. Команда `pull` по сути представляет собой комбинацию `fetch` и `merge`. То есть, используя эту единственную команду, вы можете как получить, так и объединить изменения в локальном репозитории. Например, другой разработчик, который хочет использовать обновленную кодовую базу с изменениями `feature1`, которые были объединены с основной веткой в исходном коде, может использовать команду `git pull`, чтобы `fetch` обновленную кодовую базу из исходного кода и `merge` его/ее локальную кодовую базу перед началом разработки.

Описанный здесь рабочий процесс клонирования->ветвления->слияния можно резюмировать на следующей диаграмме.



Fork

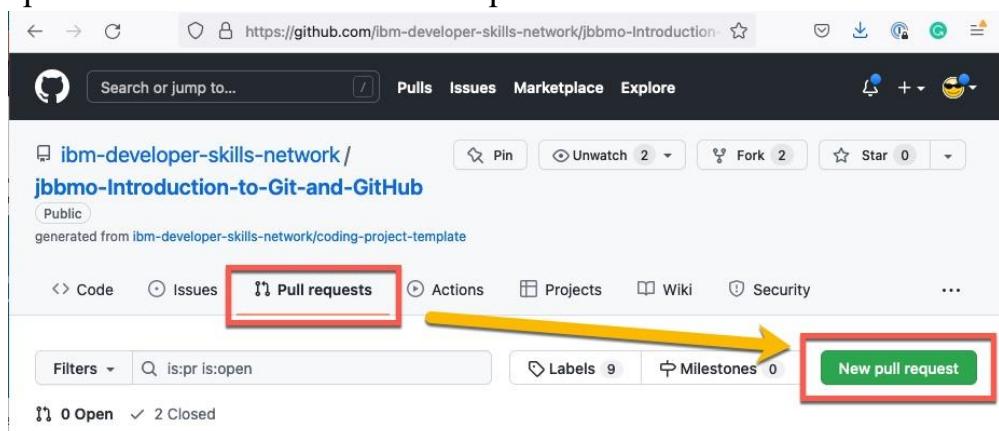
Если разработчик хочет создать производный проект с другим проектом в качестве отправной точки или работать над проектом, используя отдельный или независимый клон, он может выбрать `fork` проекта. Вы можете создать форк любого общедоступного проекта, перейдя на его страницу проекта на GitHub и нажав кнопку «`Fork`» в верхней части страницы.



ПРИМЕЧАНИЕ. Параметр *fork* доступен только с использованием веб-интерфейса, и для создания разветвления не существует команды git. Проект, из которого вы создаете ответвление (*fork*), называется upstream проектом.

После разветвления проекта разработчики, имеющие доступ к разветвлению, могут работать над обновлением и внесением изменений в разветвление, используя тот же рабочий процесс, который описан ранее, т.е. разветвленная копия проекта теперь становится источником, а разработчики, имеющие доступ к источнику, могут создавать его клоны на своих локальных машинах, где они могут создавать и объединять ветки, а также синхронизировать изменения с источником, используя pull и push.

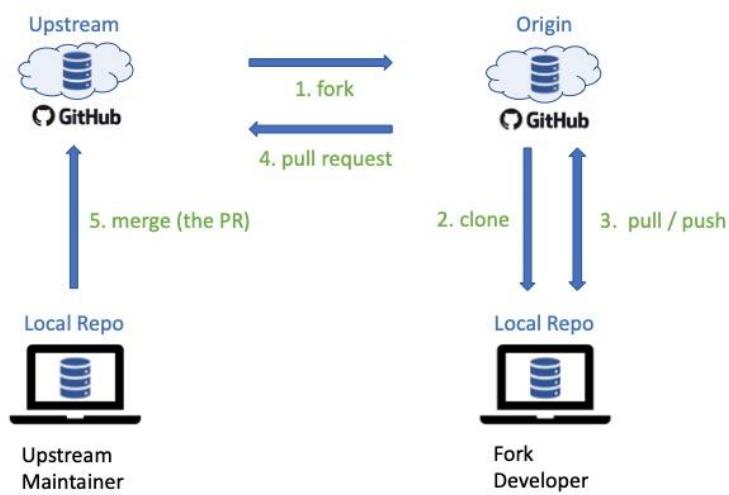
Однако важно отметить, что синхронизация изменений с использованием слияния и отправки может выполняться только с репозиториями, к которым у разработчиков есть доступ на запись, то есть в данном случае это их ответвление проекта, то есть origin, из которого они создают свои локальные клоны. Но что, если разработчик хочет внести свои изменения обратно в upstream, к которому у него нет доступа на запись? В этом случае они могут отправить pull request с предлагаемыми изменениями. Новый pull request можно создать, перейдя на домашнюю страницу проекта, перейдя на вкладку «Pull Requests» и нажав «New Pull Request».



ПРИМЕЧАНИЕ. Термин «*Pull Request*» не следует путать с командой git pull, которую вы используете для *fetch* и *merge* последней базы кода с вашим локальным репозиторием. *Pull Request*, как следует из названия, — это просто запрос на просмотр и *pull* предложенных вами изменений. В рамках PR вы

предоставляете подробную информацию о предлагаемых изменениях и вашей реализации.

Сопровождающие вышестоящего (upstream) проекта могут просмотреть изменения в PR и решить, объединять их или нет. В некоторых случаях они могут предоставить отзыв (путем комментариев в PR) или попросить отправителя PR выполнить какое-либо разрешение конфликта, например, применив свои изменения к последней кодовой базе и повторно отправив PR. Описанный здесь рабочий процесс fork->clone->pr кратко показан на рисунке ниже.



Когда форкать и клонировать?

Обычно, если у вас есть доступ к репозиторию проекта, например, как члену команды, совместно разрабатывающей кодовую базу, вы можете клонировать репозиторий и синхронизировать изменения из локальной копии репозитория, используя pull и push.

Однако, если есть общедоступный проект, в который вы хотите внести свой вклад, но у вас нет доступа для записи, или если вы используете общедоступный проект в качестве отправной точки для своего собственного проекта, вы можете разветвить проект. Затем поработайте с разветвленной кодовой базой, клонировав ее на свой компьютер и сотрудничая с командой разработчиков, работающей над разветвлением, используя синхронизацию pull-push с вашим разветвлением проекта. Но если вы хотите внести свои изменения обратно в вышестоящий проект (исходный проект, из которого вы создали ответвление), вы можете отправить свои изменения с помощью pull request – запроса на включение.

Тренировочные задания.

1. Регистрация в GitHub.

Зайдите на сайт GitHub, <https://github.com> - <https://github.com/join>

Join GitHub

Create your account

Username *

Email address *

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter.
[Learn more.](#)

Email preferences

Send me occasional product updates, announcements, and offers.

Verify your account

Please solve this puzzle so we
know you are a real person

[Verify](#)

[Create account](#)

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

Определитесь с именем пользователя, введите адрес электронной почты, придумайте пароль, нажмите кнопку «Sign up for GitHub».
Далее необходимо пройти небольшой тест, чтобы доказать, что вы человек — нажмите «Verify» и решите предоставленную головоломку.

Verify your account

Please solve this puzzle so we
know you are a real person

[Verify](#)

Затем нажмите «join a free plan», после чего вы попадете на экран, где сможете выбрать тип учетной записи — выберите бесплатную личную учетную запись (предлагается по умолчанию).

Email preferences

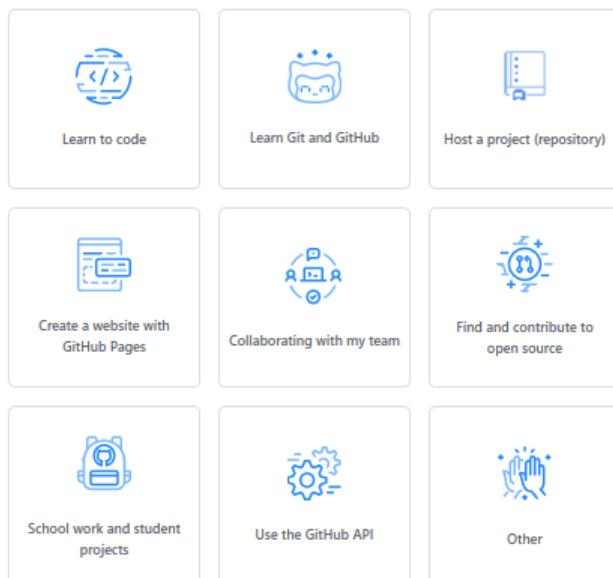
Send me occasional product updates, announcements, and offers.

[Join a free plan](#)

GitHub задает несколько вопросов о вашей работе, опыте программирования и интересах.

What do you plan to use GitHub for?

(Select up to 3)



I am interested in:

languages, frameworks, industries

We'll connect you with communities and projects that fit your interests.

For example: zeplin elm apm

[Complete setup](#)

После этого вам необходимо ответить на полученное электронное письмо, подтверждающее, что вы связались с GitHub из учетной записи, к которой вы имеете доступ.



Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address.

An email containing verification instructions was sent to [Your email address](#)

[Resend verification email](#)

[Change your email settings](#)

Далее вы можете создать репозиторий или организацию или пройти курс «Introduction to GitHub». Организация — это совокупность учетных записей пользователей, владеющих репозиториями. У организаций есть один или несколько владельцев, обладающих правами администратора для организации.

Сердцем проекта на основе Git является репозиторий. Он содержит весь ваш код и связанные с ним артефакты, включая такие вещи, как:

- Файл README, описывающий цель проекта.
- Файл LICENSE, в которой необходимо указать способы использования вашего кода иными пользователями.

Вы можете сделать свой репозиторий частным (доступным только для людей с учетными записями, имеющими разрешение на его просмотр) или общедоступным (доступным для поиска и просмотра всеми).

Когда вы создадите свой репозиторий, вы заметите, что он имеет несколько вкладок и открывается на вкладке «Code». Code – место, где находятся все исходные файлы. Git изначально создавался как хранилище исходного кода, и теперь сюда попадают самые разные файлы.

Если вы выбрали автоматическое создание README и/или LICENSE, они также будут на данной вкладке.

Issues позволяет отслеживать открытые элементы в базе вашего проекта.

Pull Requests – это часть механизма взаимодействия с другими пользователями. Pull requests определяют изменения, которые зафиксированы и готовы к проверке перед объединением в основную ветку.

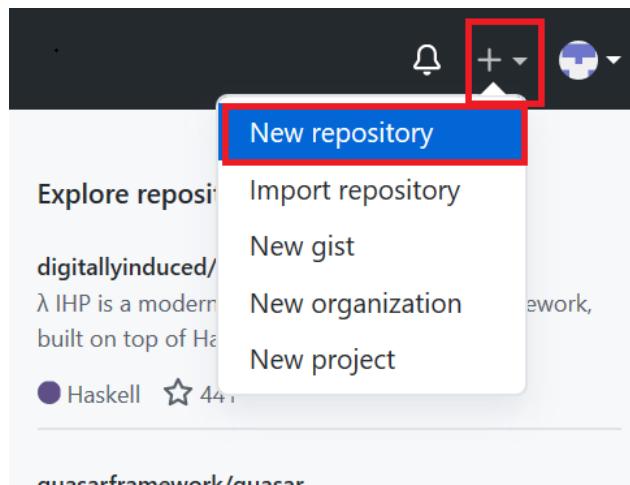
Projects – инструменты для управления, сортировки, планирования и т.д. ваших различных проектов.

Wiki, Security и Insights – эти инструменты часто оставляются для более опытных пользователей и обеспечивают основу для связи с внешним сообществом пользователей.

Settings – GitHub позволяет выполнять множество настроек, включая изменение имени вашего репозитория и контроль доступа.

2. Создание репозитория в GitHub

Нажмите «+», затем нажмите «New Repository».



Чтобы создать новый репозиторий, вам необходимо заполнить следующие данные:

- дать новому репозиторио имя;
- добавить описание репозитория;
- выбрать видимость репозитория – хотите ли вы, чтобы он был публичным или частным;
- выбрать опцию «Initialize this repository with a README».

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner * Repository name *

/

Great repository names are short and memorable. Need inspiration? How about [urban-octo-waffle](#)?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

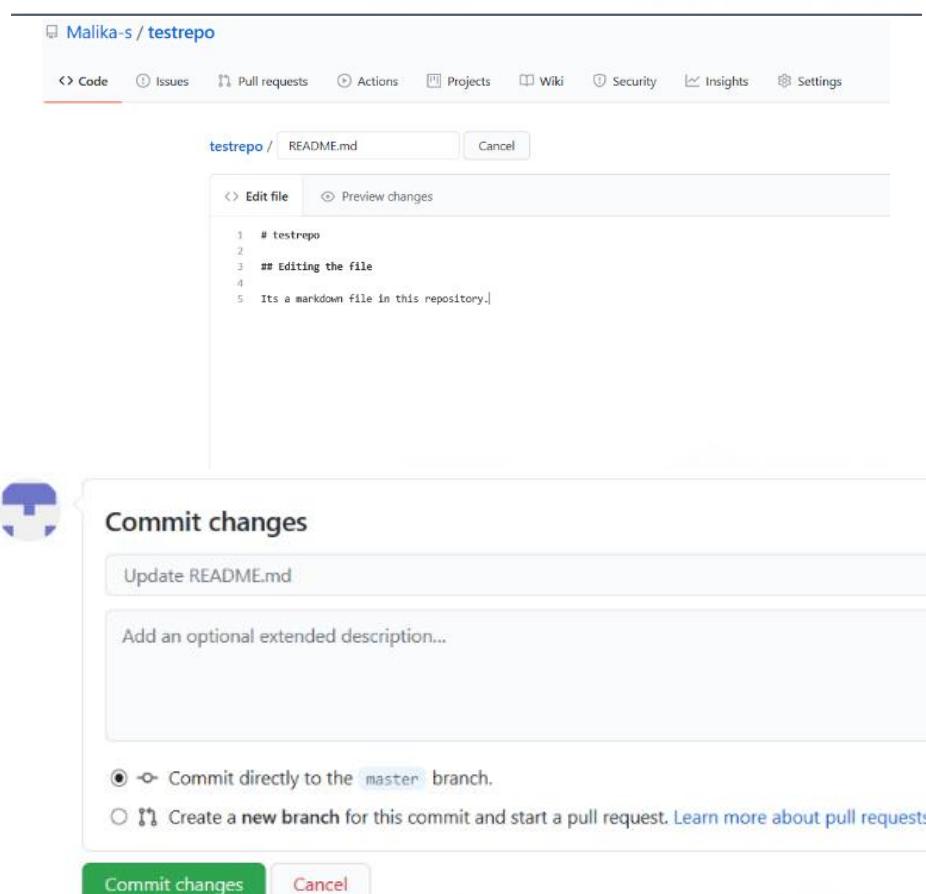
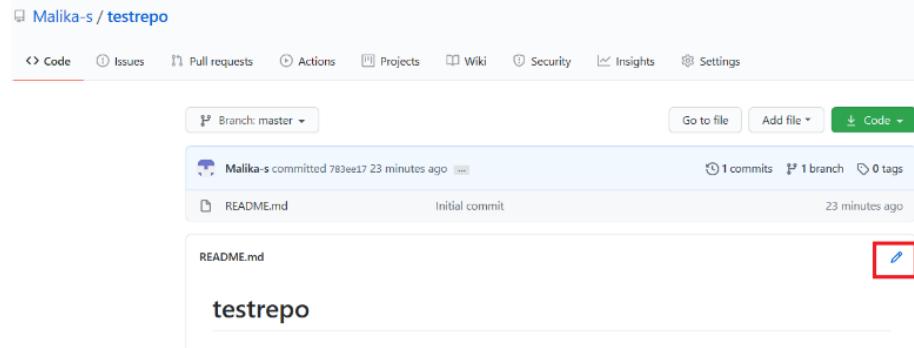
Add .gitignore: None Add a license: None

Create repository

Затем нажмите «Create Repository». Теперь вы будете перенаправлены в созданный вами репозиторий. Корневая папка вашего репозитория указана по умолчанию и содержит только один файл README.md.

Следующий шаг – отредактировать файл README. Это можно сделать в браузере: нажмите на карандаш, чтобы открыть онлайн-редактор, и вы

сможете изменить текст файла README. Чтобы сохранить изменения в репозитории, вы должны зафиксировать их. После внесения изменений прокрутите вниз до раздела «Commit changes». Добавьте сообщение о фиксации и (при необходимости) описание, затем нажмите «Commit changes». «Commit changes» используется для сохранения изменений в репозитории. Вернитесь на главный экран, щелкнув ссылку на имя репозитория. Обратите внимание, что файл readme обновлен, и проверьте внесенные изменения.



Создание нового файла с помощью встроенного веб-редактора GitHub, который запускается в браузере. Нажмите «Add File», затем нажмите «Create New File», чтобы создать новый файл. Например, создаем файл Python с именем firstpython.py. Сначала укажите имя файла. Затем добавьте комментарий, описывающий ваш код, а затем добавьте код. После завершения зафиксируйте изменения в репозитории. Вы можете видеть, что ваш файл

теперь добавлен в репозиторий, а в списке репозитория показано, когда файл был добавлен или изменен.

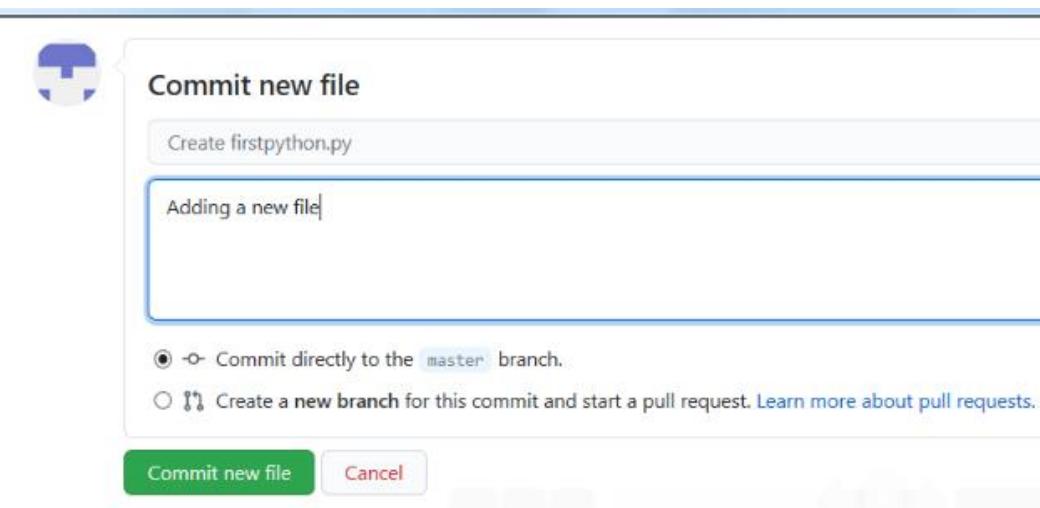
The image consists of three vertically stacked screenshots of a GitHub repository named "testrepo".

Screenshot 1: The repository page shows a single file, "README.md", which has been updated 4 minutes ago. The top navigation bar is visible, showing "Code" as the active tab.

Screenshot 2: The repository page shows the same state as Screenshot 1. A red box highlights the "Add file" button in the top right corner of the header.

Screenshot 3: The repository page shows the "Edit new file" dialog for "firstpython.py". The code editor contains the following Python script:

```
1 # Display the output
2 print("New Python file")
```



Если вам нужно изменить файл, необходимо выполнить следующее. Щелкните имя файла, а затем щелкните значок карандаша, внесите изменения и зафиксируйте их.

Вы также можете загрузить файл из вашей локальной системы в репозиторий. На главном экране репозитория нажмите «Add File» и выберите параметр «Upload files». Нажмите «Choose Your Files» и выберите файлы, которые вы хотите загрузить из своей локальной системы. Процесс загрузки файла может занять некоторое время, в зависимости от того, что вы загружаете. После завершения загрузки файлов нажмите «Commit Changes». В репозитории теперь отображаются загруженные файлы.

Malika-s / testrepo

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Branch: master

Malika-s committed 0ea980 1 hour ago

README.md Update README.md 1 hour ago

firstpython.py Create firstpython.py 1 hour ago

Code

Add file

Create new file

Upload files

Drag files here to add them to your repository

Or choose your files

The screenshot shows a GitHub commit dialog and a repository history. In the top dialog, 'Test.py' is selected, and the 'Commit changes' button is highlighted with a red box. Below the dialog, the repository history shows three commits:

File	Action	Time
README.md	Update README.md	2 hours ago
Test.py	Add files via upload	now
firstpython.py	Create firstpython.py	1 hour ago

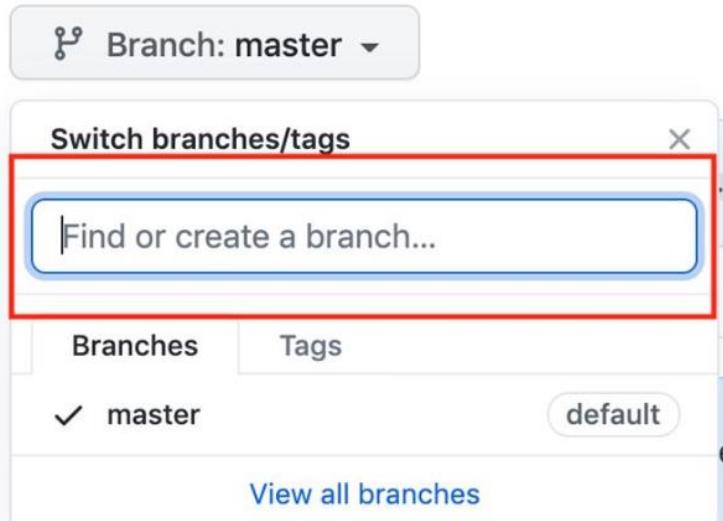
3. Создание branch

Чтобы добавить ветку в ваш репозиторий, выполните следующие шаги.

- Перейдите на главную страницу вашего репозитория. Обратите внимание, что когда вы создавали свой репозиторий, для вас была создана основная ветка.

The screenshot shows the repository history after creating a new branch. The 'Branch: master' dropdown menu is open, showing '4 commits', '1 branch', and '0 tags'. The '1 branch' option is highlighted with a red box. The history table remains the same as in the previous screenshot.

- В верхней части списка файлов найдите раскрывающееся меню «branch». (По умолчанию в меню отображается «Branch: master».) Щелкните раскрывающееся меню, введите имя ветки, которую вы хотите создать, и нажмите Enter на клавиатуре.



В вашем репозитории теперь есть две ветки: master и Child_Branch. Вы можете щелкнуть раскрывающееся меню, чтобы увидеть свои ветки.

Все файлы, которые находились в основной ветке, теперь скопированы в Child_Branch. Обратите внимание: когда вы добавляете или редактируете файл в Child_Branch, это изменение не будет автоматически внесено в основную ветку.

Чтобы добавить файл в новую ветку, убедитесь, что Child_Branch (или любое другое имя, которое вы дали своей ветке) отображается в раскрывающемся меню «Branch», и выполните следующие шаги:

- Нажмите *Add file* -> *Create new file*, чтобы создать файл в репозитории.

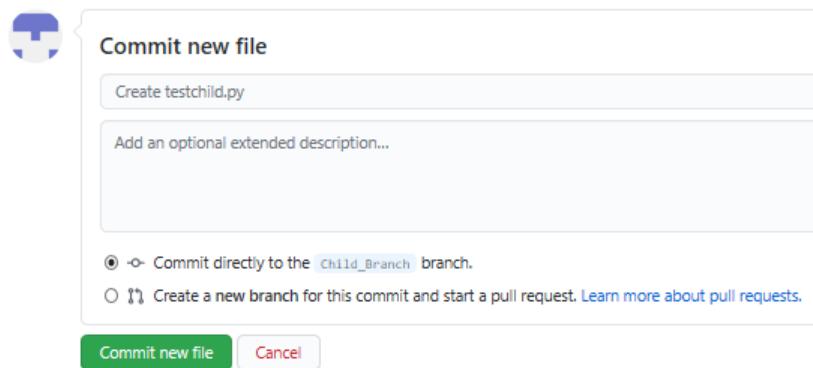
- Введите имя и расширение файла — например, testchild.py — и добавьте следующие строки в тело нового файла:

```

1 ## Adding a new file in child branch
2 print ("Inside Child branch")

```

Прокрутите страницу вниз, добавьте описание файла, который вы собираетесь добавить (обратите внимание, что описание не является обязательным), и нажмите «*Commit new file*».



Файл, который вы добавили в дочернюю ветку, не добавляется автоматически в основную ветку. (Вы можете проверить это, используя раскрывающееся меню «Branch», чтобы перейти к основной ветке; обратите внимание, что в списке файлов нет файла testchild.py)

File	Action	Count
README.md	Update	7
Test.py	Add files via upload	5
firstpython.py	Create	7

Вы также можете сравнить две ветки и открыть запрос на включение, который позволит вам скопировать изменения, внесенные вами в дочернюю ветку (в данном случае добавление нового файла), в основную ветку.

- В Child_Branch нажмите кнопку «*Compare & pull request*».

Child_Branch had recent pushes less than a minute ago

Compare & pull request

Branch: Child_Branch

This branch is 3 commits ahead of master.

Malika-s committed [ebsced2](#) 3 minutes ago

7 commits 2 branches 0 tags

File	Commit Message	Time
README.md	Update README.md	3 hours ago
Test.py	Add files via upload	1 hour ago
firstpython.py	Create firstpython.py	2 hours ago
testchild.py	Create testchild.py	3 minutes ago

Showing 1 changed file with 2 additions and 0 deletions.

```

diff --git a/testchild.py b/testchild.py
index 123456..7890ab 100644
--- a/testchild.py
+++ b/testchild.py
@@ -0,0 +1,2 @@
1 + ## Adding a new file in child branch
2 + print ("Inside Child branch")

```

Unified Split

- Прокрутите страницу вниз и обратите внимание, что в списке указан *1 changed file*.

- Прокрутите страницу вверх и обратите внимание, что GitHub сравнивает ветки master и Child_Branch и между ними нет конфликтов. При желании вы можете добавить комментарий к запросу на включение. Нажмите *Create pull request*.

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: master ▾ ← compare: Child_Branch ▾ Able to merge. These branches can be automatically merged.

Child branch

Want to change in the master branch.

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

4. Операция «Merge» на базе «pull request»

Чтобы объединить ветки по запросу pull request в проекте, выполните следующие действия:

- Откройте вкладку «Pull requests». Отображается список ожидающих запросов на включение.
- Щелкните pull request, который вы хотите объединить с основным проектом. Если вас устраивают изменения, нажмите «Merge pull request», чтобы принять запрос на включение и объединить обновления (при желании вы можете добавить комментарий).

Child branch #1

Malika-s commented 2 minutes ago
Want to change in the master branch.

Malika-s added 3 commits 25 minutes ago

- > Create testchild
- > Delete testchild
- > Create testchild.py

Verified 9767921
Verified 48b4479
Verified eb5ced2

Add more commits by pushing to the Child_Branch branch on Malika-s/testrepo.

Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

Merge pull request

- При нажатии кнопки «*Merge pull request*» отображается кнопка «*Confirm merge*». Нажмите эту кнопку, чтобы завершить слияние.

Child branch #1

Malika-s commented 4 minutes ago
Want to change in the master branch.

Malika-s added 3 commits 27 minutes ago

- > Create testchild
- > Delete testchild
- > Create testchild.py

Verified 9767921
Verified 48b4479
Verified eb5ced2

Add more commits by pushing to the Child_Branch branch on Malika-s/testrepo.

Merge pull request #1 from Malika-s/Child_Branch

Child branch

Confirm merge Cancel

pull request теперь успешно merge. Обратите внимание: вы можете удалить дочернюю ветку, поскольку ваши изменения были включены в master ветку.

Branch: master

Go to file Add file Code

Malika-s committed 7a7c576 3 minutes ago ... 8 commits 2 branches 0 tags

File	Action	Time
README.md	Update README.md	3 hours ago
Test.py	Add files via upload	1 hour ago
firstpython.py	Create firstpython.py	3 hours ago
testchild.py	Create testchild.py	21 minutes ago

5. Работа с локальным репозиторием посредством командной строки.

Данный вариант доступен либо в MacOS / Linux, либо в Windows PowerShell через специальное приложение Git.

Создаем новую директорию для локального репозитория:

- Создайте каталог myrepo, скопировав и вставив в терминал приведенную ниже команду mkdir: «mkdir myrepo».
- Перейдите в каталог myrepo, скопировав и вставив команду cd: «cd myrepo».
- В этом каталоге myrepo можно создать новый локальный репозиторий git с помощью команды git init. Скопируйте и вставьте в терминал команду: «git init»
- Теперь создан новый локальный репозиторий, который вы можете проверить, выполнив список каталогов, вставив следующую команду в окно терминала: «ls -la .git»
- Вывод показывает содержимое подкаталога .git, в котором находится локальный репозиторий:

```
theia@theiadocker-rsahuja: /home/project/myrepo ×  
  
theia@theiadocker-rsahuja: /home/project$ mkdir myrepo  
theia@theiadocker-rsahuja: /home/project$ cd myrepo  
theia@theiadocker-rsahuja: /home/project/myrepo$ git init  
Initialized empty Git repository in /home/project/myrepo/.git/  
theia@theiadocker-rsahuja: /home/project/myrepo$ ls -la .git  
total 40  
drwxr-sr-x 7 theia users 4096 Jan 15 01:53 .  
drwxr-sr-x 3 theia users 4096 Jan 15 01:53 ..  
drwxr-sr-x 2 theia users 4096 Jan 15 01:53 branches  
-rw-r--r-- 1 theia users 92 Jan 15 01:53 config  
-rw-r--r-- 1 theia users 73 Jan 15 01:53 description  
-rw-r--r-- 1 theia users 23 Jan 15 01:53 HEAD  
drwxr-sr-x 2 theia users 4096 Jan 15 01:53 hooks  
drwxr-sr-x 2 theia users 4096 Jan 15 01:53 info  
drwxr-sr-x 4 theia users 4096 Jan 15 01:53 objects  
drwxr-sr-x 4 theia users 4096 Jan 15 01:53 refs  
theia@theiadocker-rsahuja: /home/project/myrepo$ █
```

- Теперь давайте создадим пустой файл, используя следующую команду: «touch newfile».
- Добавьте этот файл в репозиторий, используя следующую команду git add: «git add newfile»
- Прежде чем мы сможем зафиксировать наши изменения, нам нужно сообщить git, кто мы такие. Мы можем сделать это, используя следующие команды (вы можете скопировать эти команды как есть, без необходимости вводить фактическую информацию):
«git config --global user.email "you@example.com"»
«git config --global user.name "Your Name"»
- Как только в репозитории появится newfile, давайте зафиксируем наши изменения, используя команду git commit. Обратите внимание, что для

фиксации требуется сообщение, которое мы включаем с помощью параметра -m: «git commit -m "added newfile"»

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git commit -m "added newfile"
[master (root-commit) 161ac8d] added newfile
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newfile
theia@theiadocker-rsahuja:/home/project/myrepo$ █
```

- Наш предыдущий коммит создал основную ветку по умолчанию под названием master. Чтобы внести последующие изменения в наш репозиторий, давайте создадим новую ветку в нашем локальном репозитории. Скопируйте и вставьте следующую команду git branch в терминал, чтобы создать ветку с именем my1stbranch: «git branch my1stbranch»
- Давайте проверим, какие ветки содержит наше репо, вставив в терминал следующую команду git branch: «git branch».

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch
* master
  my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ █
```

Обратите внимание, что в выводе перечислены две ветки – основная ветка по умолчанию со звездочкой * рядом с ней, указывающая, что это активная в данный момент ветка, и вновь созданная ветка my1stbranch.

- Поскольку теперь мы хотим работать в новой ветке, введите команду git checkout, чтобы сделать ее активной веткой для внесения изменений: «git checkout my1stbranch».
- Давайте проверим, что новая ветка теперь является активной, выполнив следующую команду git branch: «git branch»

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch
* master
  my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ git checkout my1stbranch
Switched to branch 'my1stbranch'
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch
  master
* my1stbranch
theia@theiadocker-rsahuja:/home/project/myrepo$ █
```

Обратите внимание, что звездочка * теперь находится рядом с my1stbranch, указывая, что она теперь активна.

ПРИМЕЧАНИЕ. В качестве команды для создания новой ветки и перехода к ней с помощью git, вы можете также использовать следующую команду с опцией -b: «git checkout -b my1stbranch»

- Давайте внесем некоторые изменения в вашу новую ветку под названием my1stbranch. Начните с добавления текста в newfile, вставив в терминал следующую команду, которая добавит строку «Вот текст в моем новом файле» в файл: «echo 'Here is some text in my newfile.' >> newfile»

- Убедитесь, что текст добавлен, вставив следующую команду cat: «cat newfile»

```
theia@theiadocker-rsahuja:/home/project/myrepo$ echo 'Here is some text in my newfile.' >> newfile  
theia@theiadocker-rsahuja:/home/project/myrepo$ cat newfile  
Here is some text in my newfile.
```

- Теперь давайте создадим еще один файл с именем readme.md, используя следующую команду: «touch readme.md»

- Добавьте его в репозиторий с помощью следующей команды git add: «git add readme.md»

На данный момент в нашей новой ветке мы отредактировали новый файл и добавили файл с именем readme.md. Мы можем легко проверить изменения в нашей текущей ветке с помощью команды git status. Вывод команды git status показывает, что файлы readme.md были добавлены в ветку и готовы к фиксации, поскольку мы добавили их в ветку с помощью git add. Однако, несмотря на то, что мы изменили файл с именем newfile, мы не добавили его явно с помощью git add, и, следовательно, он не готов к фиксации:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ touch readme.md  
theia@theiadocker-rsahuja:/home/project/myrepo$ git add readme.md  
theia@theiadocker-rsahuja:/home/project/myrepo$ git status  
On branch my1stbranch  
Changes to be committed:  
  (use "git reset HEAD <file>..." to unstage)  
  
    new file:   readme.md  
  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git checkout -- <file>..." to discard changes in working directory)  
  
    modified:  newfile
```

Команда для добавления всех изменений и дополнений – git add со звездочкой *; данный шаг также добавит измененный файл newfile в ветку и подготовит его к фиксации: «git add *»

- Давайте еще раз проверим статус: «git status». Вывод теперь показывает, что оба файла теперь могут быть зафиксированы:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git add *  
theia@theiadocker-rsahuja:/home/project/myrepo$ git status  
On branch my1stbranch  
Changes to be committed:  
  (use "git reset HEAD <file>..." to unstage)  
  
    modified:  newfile  
    new file:   readme.md
```

- Теперь, когда наши изменения готовы, мы можем сохранить их в ветку, используя следующую команду фиксации с сообщением, указывающим изменения: «git commit -m "added readme.md modified newfile"»

- Мы можем выполнить следующую команду git log, чтобы получить историю последних коммитов. В журнале показаны два недавних коммита — последний коммит в my1stbranch, а также предыдущий коммит в master.

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit 9eb37c754d77231a2013781aa5215f71040975ed (HEAD -> my1stbranch)
Author: Your Name <you@example.com>
Date:   Sat Jan 15 04:00:50 2022 +0000

    added readme.md modified newfile

commit 161ac8d957bd0d904c85ef8883b36c5824f10d85 (master)
Author: Your Name <you@example.com>
Date:   Sat Jan 15 02:07:41 2022 +0000

    added newfile
```

* Чтобы выйти из журнала git, введите q.

- Иногда вы не можете полностью протестировать свои изменения перед их фиксацией, и это может иметь нежелательные последствия. Вы можете отменить свои изменения, используя команду git revert, как показано ниже. Вы можете либо указать идентификатор вашего коммита, который вы можете увидеть в предыдущем выводе журнала, либо использовать ярлык HEAD для отката последнего коммита:

«git revert HEAD --no-edit»

Если вы не укажете флаг --no-edit, вам может быть представлен экран редактора, показывающий сообщение с изменениями, которые необходимо отменить. В этом случае нажмите клавишу Control (или Ctrl) одновременно с X. Вывод показывает, что самый последний коммит с указанным идентификатором был отменен:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git revert HEAD --no-edit
[my1stbranch f4f5600] Revert "modified newfile added readme.md"
Date: Sat Jan 15 04:39:38 2022 +0000
2 files changed, 1 deletion(-)
 delete mode 100644 readme.md
theia@theiadocker-rsahuja:/home/project/myrepo$ █
```

- Давайте внесем еще одно изменение в ваш текущий активный my1stbranch, используя следующие команды:

«touch goodfile»

«git add goodfile»

«git commit -m "added goodfile"»

«git log»

- Вывод журнала показывает, что недавно добавленный goodfile был зафиксирован в my1stbranch:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git status
On branch my1stbranch
nothing to commit, working tree clean
theia@theiadocker-rsahuja:/home/project/myrepo$ touch goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git add goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git commit -m "added goodfile"
[my1stbranch d8680b4] added goodfile
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit d8680b4cf63ff3e97b2970704806c14dc6134dd1 (HEAD -> my1stbranch)
Author: Your Name <you@example.com>
Date:   Sat Jan 15 04:53:27 2022 +0000

    added goodfile
```

- Теперь давайте объединим содержимое my1stbranch с основной веткой. Сначала нам нужно сделать активную ветку master с помощью следующей команды git checkout: «git checkout master»
- Теперь давайте объединим изменения из my1stbranch в master.
«git merge my1stbranch»
«git log»

Вывод и журнал показывают успешное слияние ветки:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git checkout master
Switched to branch 'master'
theia@theiadocker-rsahuja:/home/project/myrepo$ git merge my1stbranch
Updating 8954f54..d8680b4
Fast-forward
  goodfile | 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit d8680b4cf63ff3e97b2970704806c14dc6134dd1 (HEAD -> master, my1stbranch)
Author: Your Name <you@example.com>
Date:   Sat Jan 15 04:53:27 2022 +0000

    added goodfile
```

- Теперь, когда изменения были объединены в главную ветку, my1stbranch можно удалить с помощью следующей команды git branch с опцией -d: «git branch -d my1stbranch»

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch -d my1stbranch
Deleted branch my1stbranch (was d8680b4).
theia@theiadocker-rsahuja:/home/project/myrepo$ █
```

Задание №1:

- Создайте новую ветку под названием newbranch.
- Сделать новую ветку активной.
- Создайте пустой файл с именем newbranchfile.
- Добавьте вновь созданный файл в свою ветку.
- Зафиксируйте изменения в новой ветке.
- Отмените последние зафиксированные изменения.
- Создайте новый файл с именем newgoodfile.
- Добавьте последний файл в новую ветку.
- Зафиксируйте изменения.
- Объедините изменения в новой ветке с основной.

Задание №2:

Разбившись на подгруппы подвое выполнить

- fork проекта второго студента;
- clone проекта в локальный репозиторий;
- добавление и фиксирование файла;
- синхронизацию с fork-репозиторием (“git push --set-upstream origin ” + имя новой ветки, если таковая создавалась -> merge этих двух веток; если изменения выполнялись сразу в master branch, то используется простая команда «git push origin master»); для git push понадобится имя пользователя и не простой пароль пользователя, а token (classic), полученный в соответствии с инструкцией <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CD0131EN-SkillsNetwork/labs/create-personal-token/instructions.md.html> (вводится вместо пароля в ходе выполнения git push)
- формирование pull request к origin проекту на прием данного изменения.

Часть 2

Основы работы с технологиями контейнеризации Docker

Основы работы будут рассмотрены на примере создания сервера с постоянно работающим Telegram ботом.

Программа работы.

Следует изучить базовые команды ubuntu https://help.ubuntu.ru/wiki/командная_строка и рекомендуется держать данную ссылку открытой, в т.ч. использовать для формирования отчета по работе.

Создание собственного облачного сервера описано в приложении 1, однако, доступные облачные серверы не имеют обычно достаточных ресурсов для обеспечения работы значительного количества пользователей, мы его используем только как шлюз к основному рабочему серверу.

1.1. Из окна Windows PowerShell подключаемся к серверу-шлюзу по ssh

```
> ssh student@195.133.13.56  
> 24
```

1.2. От сервера-шлюза подключаемся к основному рабочему серверу
\$ ssh student@10.8.0.2

24

1.3. Создаем свой (по номеру зачетки) рабочий каталог и переходим в него

```
$ mkdir 000000 && cd 000000
```

1.4. Проверяем работу python3
\$ python3

1.5. Для того, чтобы не нарушать структуру базового python, не мешать своими установками администраторам серверов и коллегам, создаем «окружение» python env и активируем его [<https://netpoint-dc.com/blog/python-venv-ubuntu-1804/>]

```
$ python3 -m venv env
```

```
$ source env/bin/activate
```

В результате в начале командной строки появляется указание на использование окружения:

```
student@debian:~/000000$
```

становится

(env) student@debian:~/000000\$

1.6. Устанавливаем необходимые pip-пакеты, в частности, нам понадобится
\$ pip install telepot

1.7. Создаем собственную учетную запись – нового бота, как это указано в Приложении 2 (дополнительно см. "Step 2: Text /newbot to BotFather" <https://www.instructables.com/Set-up-Telegram-Bot-on-Raspberry-Pi/>).

Получаем токен, его будет достаточно для работы простейшего приложения, которое на /command1 будет отвечать Oks, а на /command2 - Ok.

```
import telepot
```

```
import time
```

```
def handle(msg):
```

```
    chat_id = msg['chat']['id']
```

```
    command = msg['text']
```

```
    print('Got command: %s' % command)
```

```
    print('From : %s' % chat_id)
```

```
    if command == '/command1':
```

```
        bot.sendMessage(chat_id, 'Oks')
```

```
    elif command == '/command2':
```

```
        bot.sendMessage(chat_id, 'Ok')
```

```
bot = telepot.Bot('***** PUT YOUR TOKEN HERE *****')
```

```
bot.message_loop(handle)
```

```
print('I am listening ...')
```

```
while 1:
```

```
    time.sleep(10)
```

Копируем текст программы в буфер обмена, запускаем текстовый редактор

```
$ nano bot.py
```

Нажимаем в открывшемся пустом документе правой кнопкой мыши, текст программы вставляется.

Клавишами доходим до текста

***** *PUT YOUR TOKEN HERE* *****

удаляем его, копируем токен, полученный от BotFather на его место.

Сохраняем файл (Ctrl+O), выходим (Ctrl+X).

Запускаем программу

```
$ python3 bot.py
```

Проверяем работу бота, находя его через telegram.

1.8. Настраиваем работу собственной python-программы в виде docker-контейнера с автозапуском после старта ОС [в соответствии с <https://habr.com/ru/companies/vdsina/articles/555540/> и <https://habr.com/ru/companies/ruvds/articles/439980/>]

1.8.1. Создаем файл requirements.txt со списком pip-библиотек, необходимых для работы нашей программы

```
$ nano requirements.txt
```

telepot==12.7

Сохраняем файл (Ctrl+O), выходим (Ctrl+X).

1.8.2. Создаем файл для сборки docker образа

```
$ nano Dockerfile
```

Вставляем в файл

FROM python:3.10 AS builder

COPY requirements.txt .

RUN pip install --user -r requirements.txt

FROM python:3.10-slim

WORKDIR /code

COPY --from=builder /root/.local /root/.local

COPY ./bot.py .

```
ENV PATH=/root/.local:$PATH
```

```
CMD [ "python", "-u", "./bot.py" ]
```

Сохраняем файл (Ctrl+O), выходим (Ctrl+X).

Следует обратить внимание на версию python: указанная в данном примере соответствует python, установленном на сервере 10.8.0.2, если Вы используете свой сервер, но следует указать версию, установленную на нем, например, для Debian 12 по умолчанию устанавливается python:3.11

1.8.3. Собираем docker образ с именем – номером зачетки
\$ docker build -t 000000 .

1.8.4. Запускаем docker образ в режиме работы в фоне (-d) и даем команду запуска при перезапуске docker (--restart=always)

```
$ docker run -d --restart=always 000000
```

В ответ на данную команду, docker сообщает CONTAINER ID вида

```
5df687ebc2f6380abd23e4ac5f7899c5f9a8a0e414cfa633ffefb0c372e40fcd
```

Также данный CONTAINER ID можно понять из списка, выдаваемого в ответ на команду

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
5df687ebc2f6	000000	"python -u ./bot.py"	About a minute ago	Up About a minute		bold_bhabha

В данном случае CONTAINER ID значительно короче; можно пользоваться любым номером.

Например, для просмотра log'ов (в данном случае - результатов работы функций "print()" программы).

```
$ docker logs 5df687ebc2f6380abd23e4ac5f7899c5f9a8a0e414cfa633ffefb0c372e40fcd
```

```
Got command: /command1
```

```
From : 384540256
```

1.8.5. После проверки следует сохранить docker image в виде архива. Это может быть полезно для передачи Вашим заказчикам, например, если нет желания и возможности воспользоваться Docker Hub

Сначала останавливаем контейнер

```
$ docker stop 5df687ebc2f6
```

Далее сохраняем командой [https://stackoverflow.com/questions/24482822/how-to-share-my-docker-image-without-using-the-docker-hub]

```
$ docker save -o <path for created tar file> <image name>
```

Например,

```
$ docker save -o ./docker_image_000000.tar 000000
```

Далее из новой Windows PowerShell копируем этот файл к себе на ПК командой scp [https://wiki.enchtex.info/tools/console/scp], выполнив на локальном ПК

```
> scp student@195.133.13.56:/home/student/000000/docker_image_000000.tar .
```

(необходимо будет указать пароль пользователя - 24)

Если Вы наоборот, создавали docker на своем ПК и хотите передать его на какой-либо сервер, то команда должна иметь вид

```
$ scp ./docker_image_000000.tar student@195.133.13.56:/home/student/000000/
```

На целевом для запуска контейнера ПК распаковываем tar-файл в каталог docker

```
$ docker load -i ./docker_image_000000.tar
```

```
571ade696b26: Loading layer  
[=====] 77.82MB/77.82MB
```

```
e96fe707bd25: Loading layer  
[=====] 9.551MB/9.551MB
```

```
4b58789a003f: Loading layer  
[=====] 35.28MB/35.28MB
```

```
2655468a1a9f: Loading layer  
[=====] 5.12kB/5.12kB
```

```
0b61847d9826: Loading layer  
[=====] 12.99MB/12.99MB
```

```
801e13511f2f: Loading layer  
[=====] 1.536kB/1.536kB
```

```
808b39b933eb: Loading layer  
[=====] 10.97MB/10.97MB
```

```
ea286d404790: Loading layer  
[=====] 3.072kB/3.072kB
```

```
Loaded image: 000000:latest
```

Запускаем образ

```
$ docker run -d --restart=always 000000
```

```
50046704457e9745897ba2c36e99e9c115ef89f3c41fa443beca5a7668668342
```

Дополнительные примеры команд:

Остановка

```
$ docker stop 50046704457e9745897ba2c36e99e9c115ef89f3c41fa443beca5a7668668342
```

```
50046704457e9745897ba2c36e99e9c115ef89f3c41fa443beca5a7668668342
```

Удаление котейнера

```
$ docker rm 50046704457e9745897ba2c36e99e9c115ef89f3c41fa443beca5a7668668342
```

Удаление образа image

```
$ docker image rm 000000
```

Untagged: 000000.latest

Deleted: sha256:d863657cdb75bd42ae87353533ab6bd2201cfce16a7be2ec3c7b028b6341e5ab

Deleted: sha256:302c36b8d0f3add37345463c5235c21088c047fa0d6f19693bd9f66db7442bc6

Deleted: sha256:e87a9bf251080da4af8e7ed51e4048a453b1deb7f9d73a526b08f96f343b366b

Deleted: sha256:8466b1a13177626112a3b27b33bb6dd8bb013e4bdedf27c2d29c52b31c26d908

Deleted: sha256:4ec2e811eb3bca56124349186481b085bfa8f26b073777fcf94af8b4ac785649

Deleted: sha256:e8f29a68c53010eb307c1a07df90028fe4b7e0766bd9d7be0fc80287fa93a116

Deleted: sha256:e2b880ce8192f572ac31d7db0071633982c965863927fe8e5048d32f88cc6218

Deleted: sha256:cd6b0451a6fea2b8ab85302915856393122ec8db6399363a01d273a98c819ac2

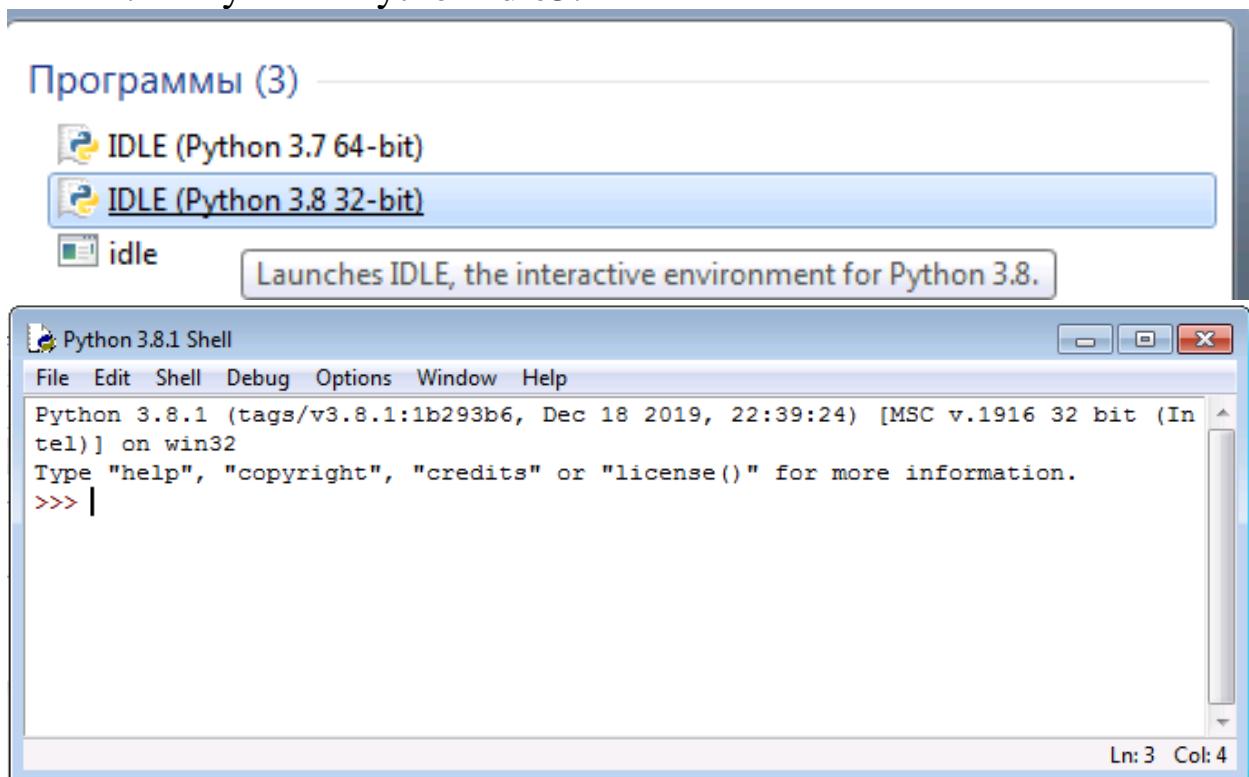
Deleted: sha256:571ade696b261f0ff46e3cdac4635afc009c4ed3429950cb95cd7e5f70ba0a07

Часть 3

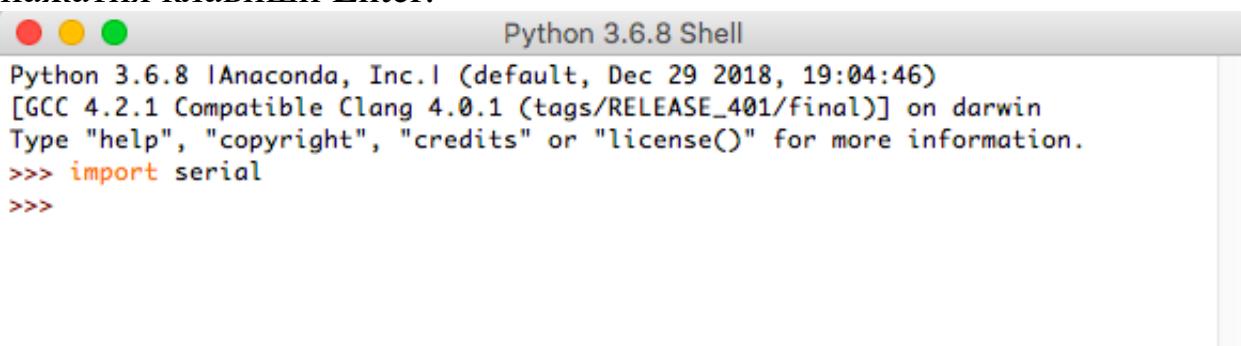
Реализация программы работы с последовательным портом средствами Python

Пояснения

0. Подготовка к работе: изучить https://en.wikipedia.org/wiki/Serial_port
1. Установить среду программирования Python (<https://www.python.org/downloads/>).
 2. Запустить Python Idle3.



3. Проверить наличие необходимой библиотеки *pyserial*.
Обратите внимание, что в данной среде команды выполняются по одной (Python – интерпретируемый язык программирования), после нажатия клавиши Enter.



4. Возникновение ошибки свидетельствует об отсутствии соответствующей библиотеки в текущей версии Python. Для установки:

4.1. Закрыть Python Idle

4.2. Запустить командную строку ОС.

4.4. Установить библиотеку pyserial для чего набрать команду

pip3 install pyserial

или

pip install serial

или

python -m pip install pyserial

5. Повторить с п. 2. В случае повторения ошибки изучить

<https://pyserial.readthedocs.io/en/latest/pyserial.html> и реализовать.

6. Создать новую многострочную программу Python (Python Ilde – File – New File).

7. Опробовать программу со следующим текстом.

```
import serial
import time
import serial.tools.list_ports
speeds = ['1200', '2400', '4800', '9600', '19200', '38400', '57600', '115200']
ports = [p.device for p in serial.tools.list_ports.comports()]
port_name = ports[0]
port_speed = int(speeds[-1])
port_timeout = 10
ard = serial.Serial(port_name, port_speed, timeout = port_timeout)
time.sleep(1)
ard.flushInput()
try:
    msg_bin = ard.read(ard.inWaiting())
    msg_bin += ard.read(ard.inWaiting())
    msg_bin += ard.read(ard.inWaiting())
    msg_bin += ard.read(ard.inWaiting())
    msg_str_ = msg_bin.decode()
    print(len(msg_bin))
except Exception as e:
    print('Error!')
ard.close()
time.sleep(1)
```

```
print(msg_str_)
```

8. Подготовить комментарии к каждой строке программы.
9. Упаковать программу в Docker контейнер по аналогии с ч. 2.
10. Разместить проект в собственном репозитории Github по аналогии с ч. 1.

КОНТРОЛЬНЫЕ ВОПРОСЫ

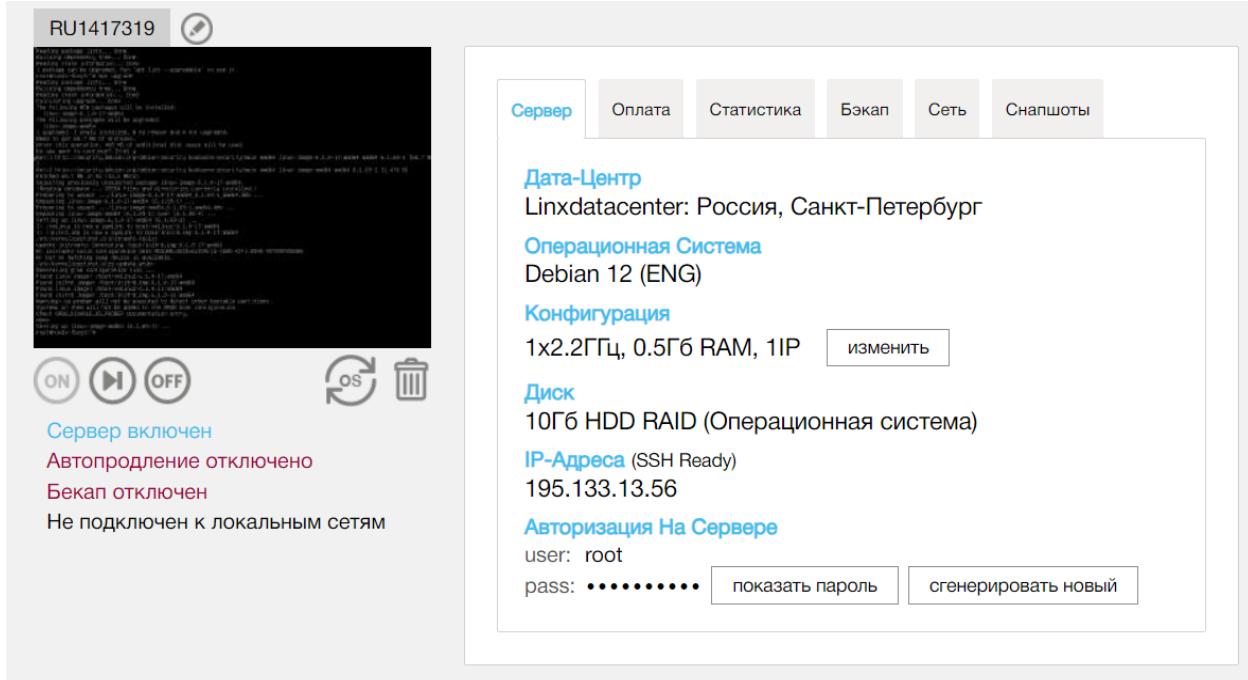
1. Что такое последовательный порт и для чего он в настоящий момент используется?
2. Какие версии Python поддерживают работу с pyserial?

Приложение 1

Подготовка собственного облачного сервера для выполнения программ в docker

1.1. Покупаем себе сервер VPS, например, вот тут <https://ruvds.com/ru-rub/my/orders>, выбирая в качестве ОС, например, Debian 12.

1.2. Ждем, пока завершится установка, видим в <https://ruvds.com/ru-rub/my/servers> новый сервер, его IP, просматриваем и копируем пароль.



1.3. На рисунке показан существующий сервер 195.133.13.56 и его можно использовать. Из Windows PowerShell подключаемся к нему удаленно под пользователем root

> ssh root@195.133.13.56

1.4. Стандартные команды проверки последних обновлений для Ubuntu/Debian после установки:

apt-get update

apt-get upgrade

Кроме того, устанавливаем htop

apt install htop

1.5. Создаем нового пользователя student с собственным каталогом

useradd -m student -s /bin/bash

и задаем ему пароль

passwd student

24

24

1.6. Добавляем его в группу, которая может подключаться по ssh к серверу

[<https://ostechnix.com/allow-deny-ssh-access-particular-user-group-linux/>]

nano /etc/ssh/sshd_config

в конце файла добавляем

AllowUsers student

Сохраняем файл (Ctrl+O), выходим (Ctrl+X).

Перезапускаем службу ssh

systemctl restart sshd

Пробуем из второго окна Windows PowerShell подключиться с указанными учетными данными

> ssh student@195.133.13.56

> 24

* Иногда проявляется ошибка в подключении к серверу по ssh (долгое ожидание сообщение о невозможности подключения). В таком случае следует воспользоваться
<https://serverfault.com/a/918810> https://www.seei.biz/ssh-fails-to-connect-with-debug1-expecting-ssh2_msg_kex_ecdh_reply/

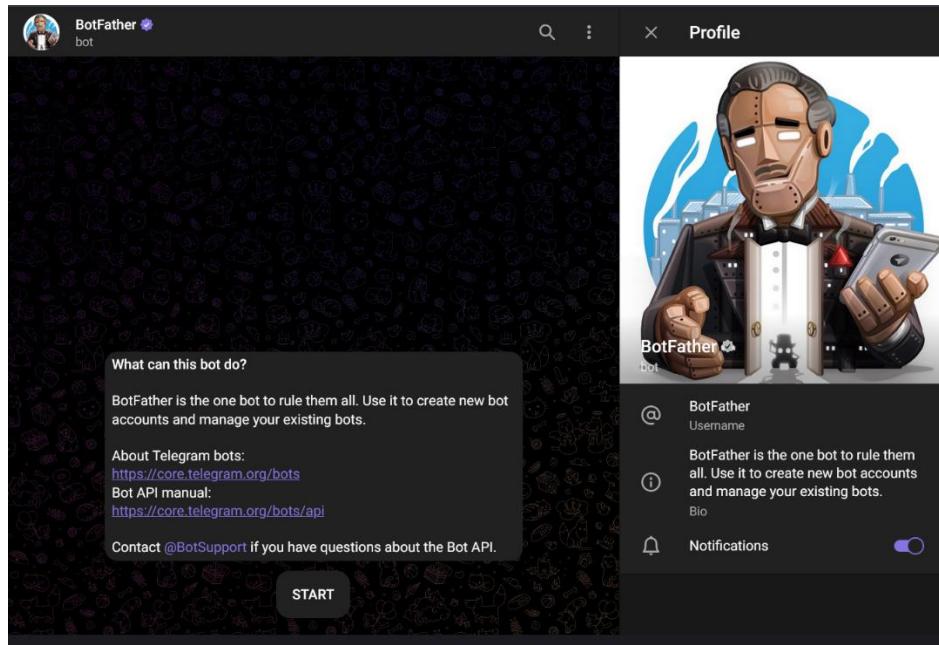
1.7. В первом окне Windows PowerShell из-под учетной записи root устанавливаем docker engine <https://docs.docker.com/engine/install/debian/>

```
# apt-get install ca-certificates curl
# install -m 0755 -d /etc/apt/keyrings
# curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
# chmod a+r /etc/apt/keyrings/docker.asc
# echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
# apt-get update
# apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Проверяем работу под root (после указанной команды ошибок быть не должно)
# docker run hello-world
Добавляем группу docker
# groupadd docker
Добавляем в эту группу student'a
# usermod -aG docker student
В окне с учетной записью student сначала переобновляем свои данные в группе
$ newgrp docker
потом проверяем работу (после указанной команды ошибок быть не должно)
$ docker run hello-world

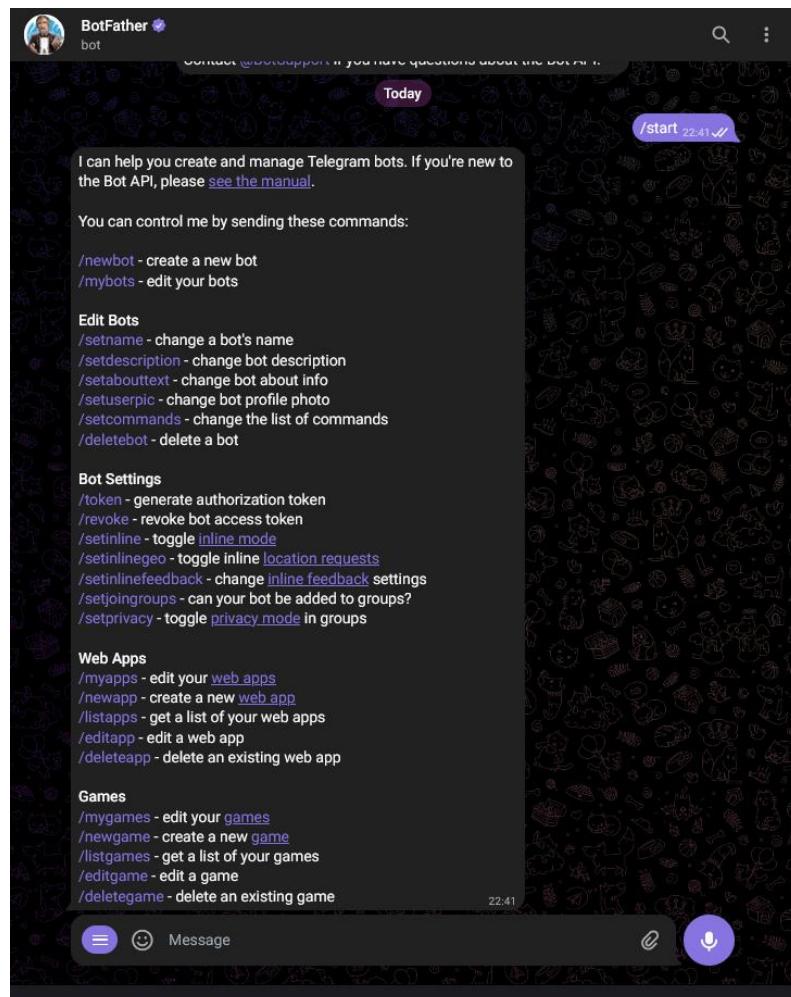
1.8. Устанавливаем python из учетной записи root
# apt install python3 python3-pip python3-venv
Затем снимаем запрет student'у устанавливать пакеты через pip
# rm /usr/lib/python3.11/EXTERNALLY-MANAGED
```

Приложение 2

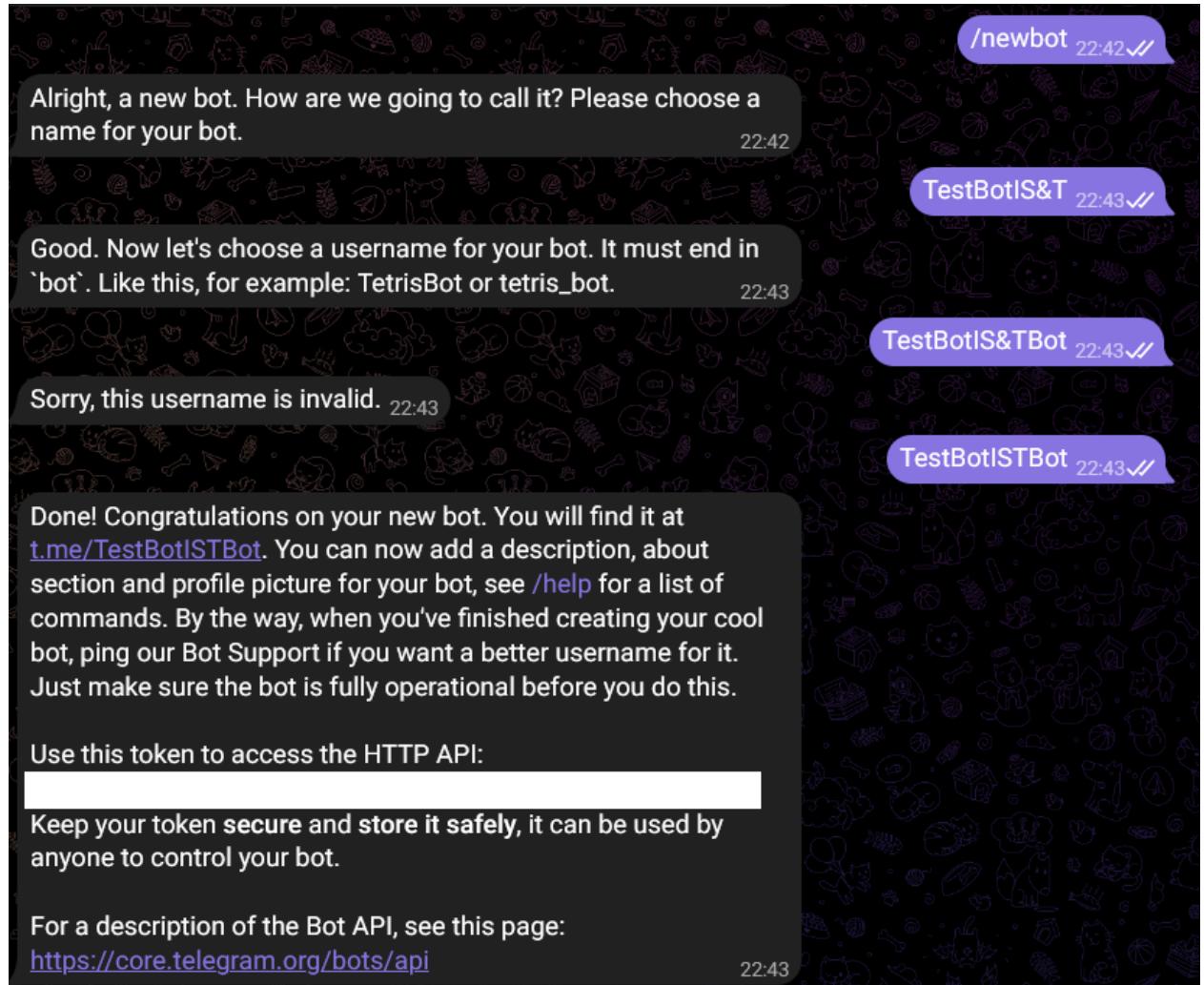
Откройте Telegram на своем телефоне, найдите пользователя по имени BotFather (поиск формирует несколько аналогов, нужен именно @BotFather). Как следует из названия, он является «отцом» всех ботов.



Он принимает специальные команды.



Чтобы получить учетную запись бота, отправьте ему сообщение /newbot. Он задаст пару вопросов. В конце процесса вам будет предоставлен токен, имеющий вид аналогичный 123456789:ABCdefGhIJKLMNOPQRSTUVWXYZ. Этот токен вам потребуется для работы программы на python. Скопируйте его в текстовый файл.



Практическая работа №2

Предварительная обработка измерительных данных в среде LabVIEW

Цель работы: получить навыки реализации программы интеллектуальной обработки данных в среде LabVIEW и, при этом, используя широкие возможности языка программирования Python.

Подготовка к работе заключается в самостоятельном изучении открытых источников по теме занятия, в частности <https://www.ni.com/en/support/documentation/supplemental/18/installing-python-for-calling-python-code.html>

Работа предусматривает подробное изучение средств предварительной разработки программ Python Anaconda и Jupyter Notebook, а также создание приложений обработки данных алгоритмами

- Линейной регрессии (Simple Linear Regression).
- Множественной линейной регрессии (Multiple Linear Regression).
- Нелинейного регрессионного анализа (Non Linear Regression Analysis).

Часть 1

Основы Anaconda и Jupyter Notebook

Одним из наиболее распространенных средств создания AI-систем является язык программирования Python. Стоит учитывать наличие значительного количества сред программирования для данного ЯП. Наиболее распространенные:

- Python Idle – стандартная среда программирования, не требует значительных ресурсов, используется для отладки программ «на месте» (например, в различных micro-PC таких как Raspberry Pi, Orange Pi и т.д.);

- Spyder – более функциональная среда по сравнению с Python Idle, используется для разработки и первично отладки программ на языке Python;

- Jupyter Notebook – среда разработки, апробации и первичной отладки алгоритмов интеллектуальной обработки данных (например, апробации различных вариантов искусственных НС для последующего встраивания их в общую программу).

Для использования всех указанных выше инструментов необходимо установить дистрибутив языков программирования Python и R, включающий набор популярных свободных библиотек, объединённых проблематиками науки о данных и машинного обучения (<https://www.anaconda.com/download/#windows> или google -> anaconda download).

Процесс установки достаточно прост и не отличается от установки любой другой программы. После установки в списке программ появится Anaconda Navigator (рис.).

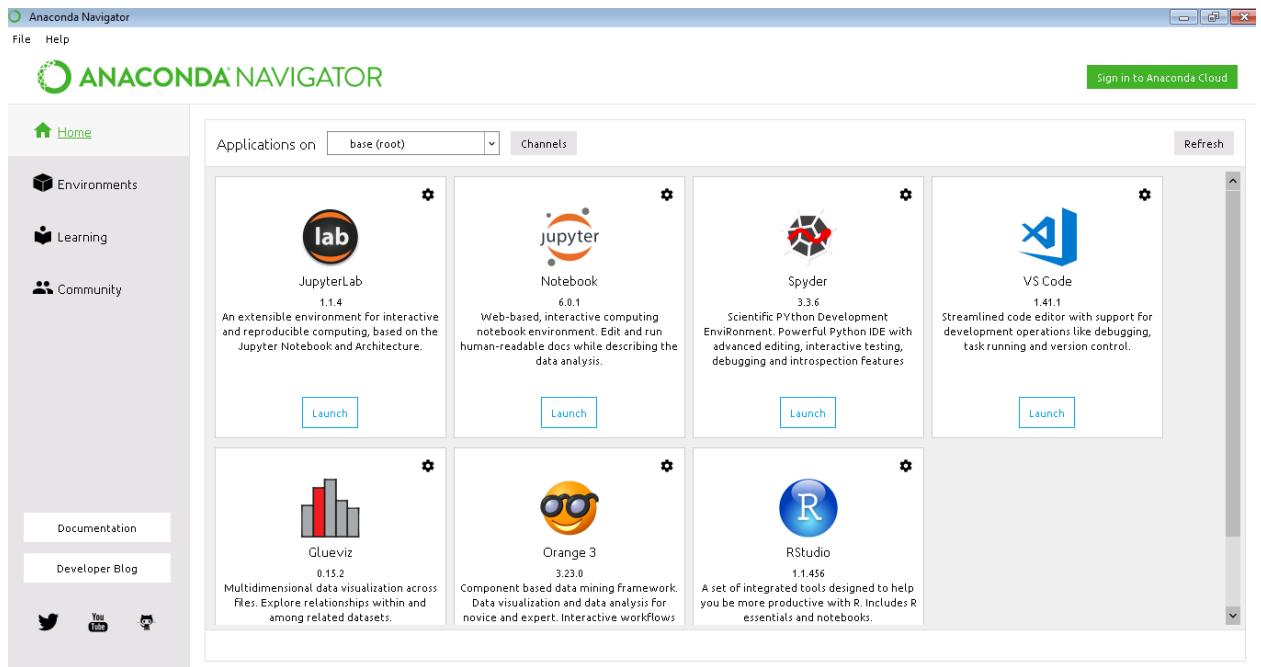


Рисунок – Anaconda Navigator

Запускаем jupyter Notebook (“Launch”) – интерфейсом программирования будет Ваш браузер (рис.).



Рисунок – Jupyter Notebook (директории зависят от Вашего компьютера)

1. В правом верхнем углу нажать – “New” → “Folder”. В списке появится “Untitled Folder” (папка без названия).
2. Поставить галочку слева от имени “Untitled Folder”.
3. В левом верхнем углу появится кнопка “Rename” (переименовать) → вводим имя каталога (work1).
4. Нажимаем на новый каталог.
5. “New” → “Python 3”. Открывается новая вкладка браузера с файлом “Untitled” (.ipynb). Закрываем вкладку.
6. В каталоге work1 видим файл Untitled.ipynb ставим возле него галочку → нажимаем на появившуюся вверху слева кнопку “Shutdown” (остановить выполнение).

7. Снова файл Untitled.ipynb ставим возле него галочку → нажимаем на появившуюся вверху слева кнопку “Rename” → вводим имя файла (work1.ipynb).

8. Нажимаем на файл work1.ipynb открывается окно в новой вкладке браузера (рис.).

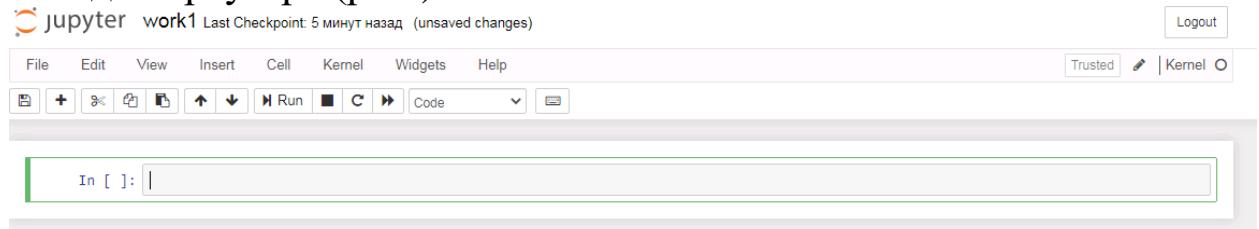


Рисунок – Jupyter Notebook work1.ipynb

8. Текст программ в Jupyter Notebook разделен на блоки. Каждый блок выполняется отдельно – для этого нажимается сочетание клавиш Shift + Enter (рис.). После выполнения блока ему присваивается номер (“In [1]”) и автоматически создается новый блок (также его можно создать кнопкой “+” в верхней панели инструментов). Результат сохраняется в оперативной памяти.

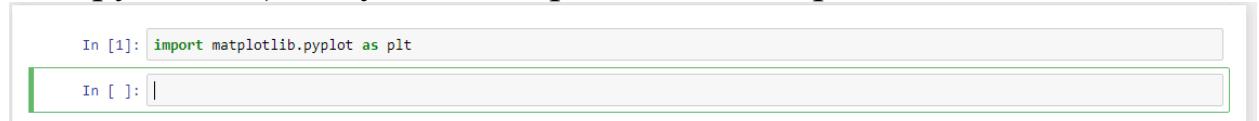


Рисунок – Jupyter Notebook work1.ipynb

9. Можно вернуться к выполненному блоку и выполнить его повторно (номер блока инкрементируется, рис.).

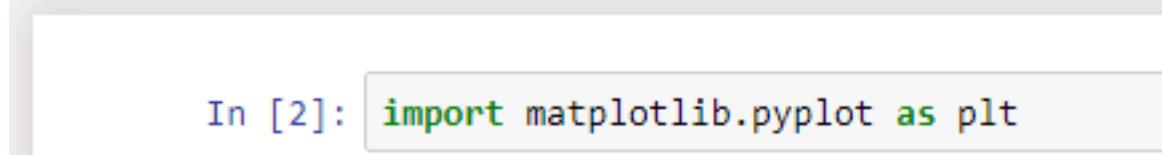


Рисунок – Jupyter Notebook work1.ipynb

10. Очистка оперативной памяти для данной программы производится перезагрузкой ядра (кнопка  на панели инструментов).

11. Разделение на блоки удобно, когда требуется провести анализ обработки одних и тех же данных различными алгоритмами (рис.).

12. Необходимо проверить наличие необходимых библиотек (рис.).

```
In [1]: import numpy as np # импорт библиотеки нужен, чтобы можно было работать с ее функциями
# импортируется каждая библиотека отдельно, чтобы все не занимали слишком много ОЗУ
# библиотека питрп используется для быстрой работы с числами и массивами чисел
# as np - далее в тексте программы будем ссылаться на эту библиотеку через имя np
# np - общепринятый акроним питрп
```

```
In [2]: x = np.array([[1, 2, 3], [4, 5, 6]]) # создаем питрп-массив 2x3
```

```
In [3]: print("x:\n{}".format(x)) # выводим значения массива с отделением его имени от значений знаком \n - перенос на новую строку
```

```
x:
[[1 2 3]
 [4 5 6]]
```

```
In [4]: print("x-array:{}\n".format(x)) # выводим значения массива без отделением его имени от значений
```

```
x-array: [[1 2 3]
 [4 5 6]]
```

```
In [ ]:
```

Рисунок – Jupyter Notebook work1.ipynb

```
In [1]: import numpy as np # work with numbers
import pandas as pd # work with various data
import matplotlib.pyplot as plt # work with graphs
import sklearn as skl # machine learning library|
```

Рисунок – Проверка наличия библиотек (должно выполниться без ошибок)

```
In [2]: arr = np.random.rand(15,2) # создаем массив случайных чисел - 15x2
```

```
In [3]: plt.figure(figsize=(20,5)) # создаем область для рисования размером 20x5
plt.xlabel('point number') # задаем название горизонтальной оси
plt.ylabel('point value') # задаем название вертикальной оси
plt.plot(arr) # выводим графики на экран
```

```
Out[3]: [<matplotlib.lines.Line2D at 0xa146352e8>,
<matplotlib.lines.Line2D at 0xa146354e0>]
```

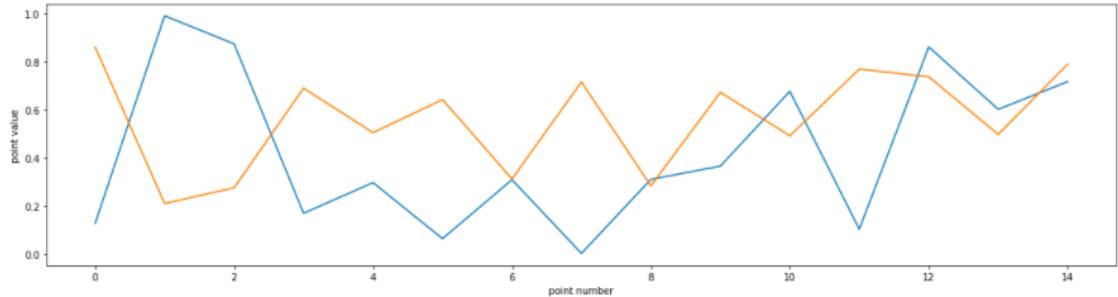


Рисунок – Программа для выполнения и отчета

Часть 2

Simple Linear Regression

Теоретические сведения

In this notebook, we learn how to use scikit-learn to implement simple linear regression. We download a dataset that is related to fuel consumption and Carbon dioxide emission of cars. Then, we split our data into training and test sets, create a model using training set, Evaluate your model using test set, and finally use model to predict unknown value

В этой практической работе мы узнаем, как использовать scikit-learn для реализации простой линейной регрессии. Мы загружаем набор данных, который связан с расходом топлива и выбросами углекислого газа автомобилей. Затем мы разделяем наши данные на обучающие и тестовые наборы, создаем модель с использованием обучающего набора, оцениваем вашу модель с помощью тестового набора и, наконец, используем модель для прогнозирования неизвестного значения.

Importing Needed packages

```
In [1]: import matplotlib.pyplot as plt  
import pandas as pd  
import pylab as pl  
import numpy as np  
%matplotlib inline
```

Understanding the Data

FuelConsumption.csv :

We have downloaded a fuel consumption dataset, **FuelConsumption.csv**, which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada. [Dataset source](#)

- **MODELYEAR** e.g. 2014
- **MAKE** e.g. Acura
- **MODEL** e.g. ILX
- **VEHICLE CLASS** e.g. SUV
- **ENGINE SIZE** e.g. 4.7
- **CYLINDERS** e.g 6
- **TRANSMISSION** e.g. A6
- **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 9.9
- **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 8.9
- **FUEL CONSUMPTION COMB (L/100 km)** e.g. 9.2
- **CO2 EMISSIONS (g/km)** e.g. 182 --> low --> 0

Reading the data in

```
In [2]: df = pd.read_csv("FuelConsumption.csv")  
  
# take a look at the dataset  
df.head()
```

```
Out[2]:
```

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FL
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	

Data Exploration

Lets first have a descriptive exploration on our data.

```
In [3]: # summarize the data  
df.describe()
```

```
Out[3]:
```

	MODELYEAR	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUEL
count	1067.0	1067.000000	1067.000000	1067.000000	1067.000000	1067.000000
mean	2014.0	3.346298	5.794752	13.296532	9.474602	
std	0.0	1.415895	1.797447	4.101253	2.794510	
min	2014.0	1.000000	3.000000	4.600000	4.900000	
25%	2014.0	2.000000	4.000000	10.250000	7.500000	
50%	2014.0	3.400000	6.000000	12.600000	8.800000	
75%	2014.0	4.300000	8.000000	15.550000	10.850000	
max	2014.0	8.400000	12.000000	30.200000	20.500000	

Lets select some features to explore more.

```
In [4]: cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]  
cdf.head(9)
```

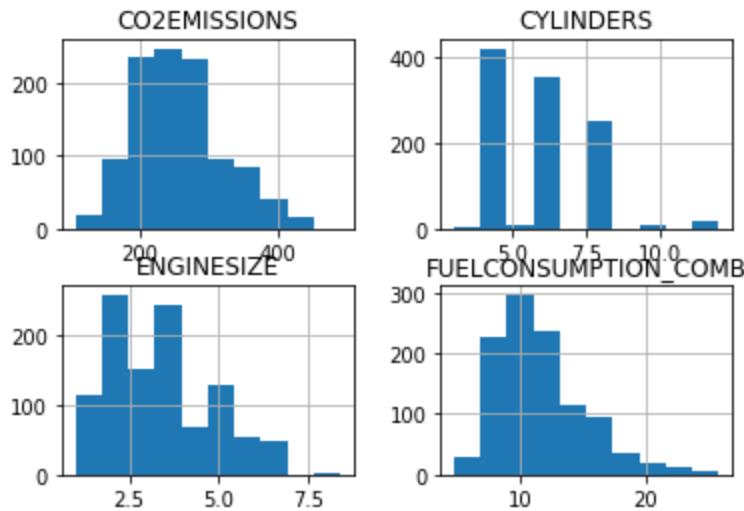
```
Out[4]:
```

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230

6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267

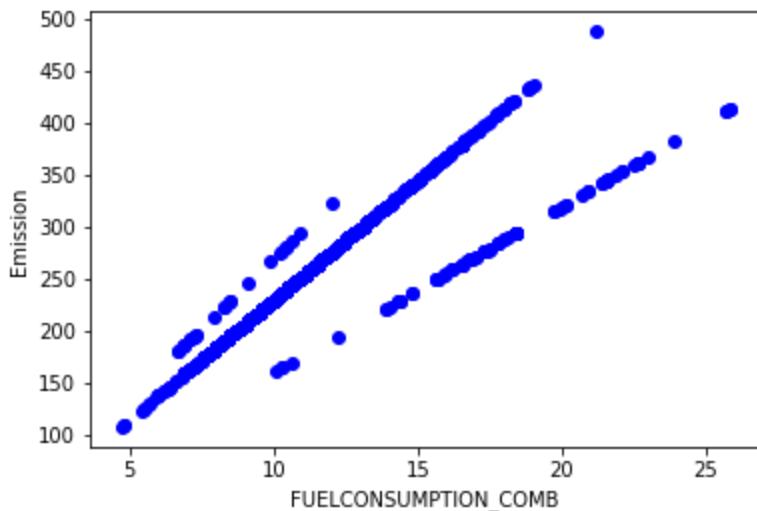
we can plot each of these features:

```
In [5]: viz = cdf[['CYLINDERS','ENGINESIZE','CO2EMISSIONS','FUELCONSUMPTION_COMB']]
viz.hist()
plt.show()
```

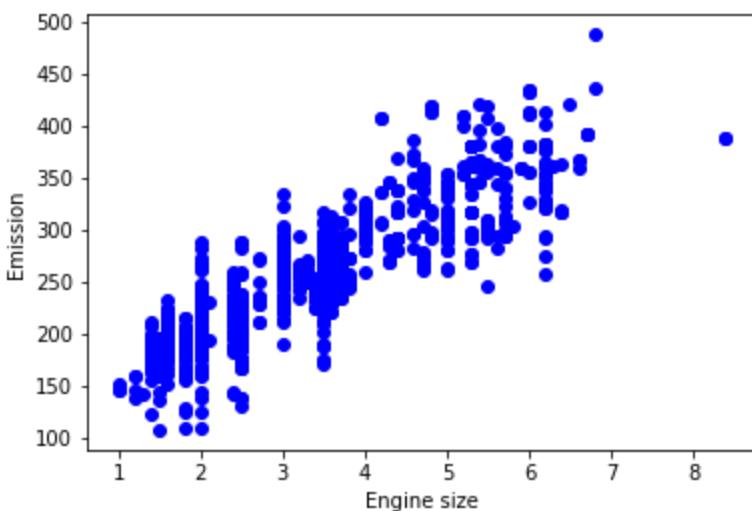


Now, lets plot each of these features vs the Emission, to see how linear is their relation:

```
In [6]: plt.scatter(cdf.FUELCONSUMPTION_COMB, cdf.CO2EMISSIONS, color='blue')
plt.xlabel("FUELCONSUMPTION_COMB")
plt.ylabel("Emission")
plt.show()
```



```
In [7]: plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS, color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



Practice

plot **CYLINDER** vs the Emission, to see how linear is their relation:

In [8]: `# write your code here`

Double-click **here** for the solution.

Creating train and test dataset

Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the data. It is more realistic for real world problems.

This means that we know the outcome of each data point in this dataset, making it great to test with! And since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. So, in essence, it is truly an out-of-sample testing.

Lets split our dataset into train and test sets, 80% of the entire data for training, and the 20% for testing. We create a mask to select random rows using **np.random.rand()** function:

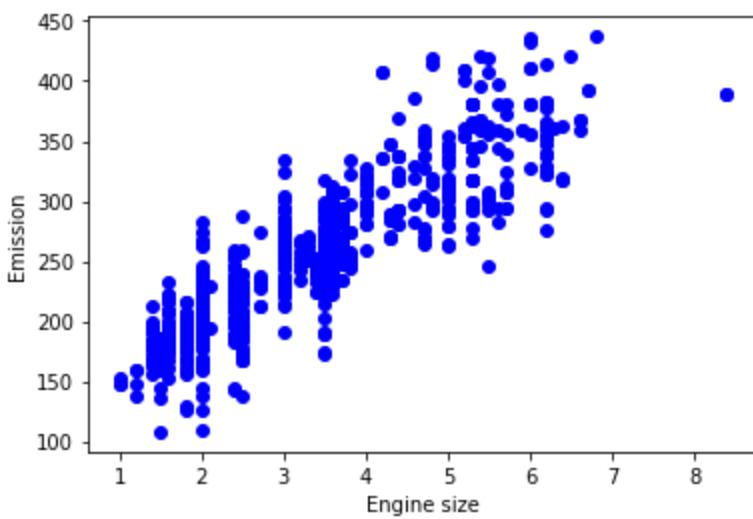
In [9]: `msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]`

Simple Regression Model

Linear Regression fits a linear model with coefficients $B = (B_1, \dots, B_n)$ to minimize the 'residual sum of squares' between the independent x in the dataset, and the dependent y by the linear approximation.

Train data distribution

In [10]: `plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()`



Modeling

Using sklearn package to model data.

```
In [11]: from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asarray(train[['ENGINESIZE']])
train_y = np.asarray(train[['CO2EMISSIONS']])
regr.fit (train_x, train_y)
# The coefficients
print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

```
Coefficients:  [[39.01008445]]
Intercept:  [125.25199996]
```

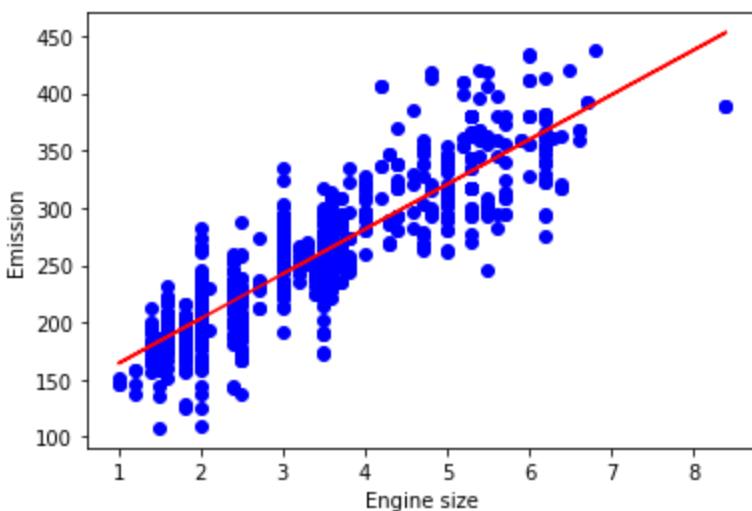
As mentioned before, **Coefficient** and **Intercept** in the simple linear regression, are the parameters of the fit line. Given that it is a simple linear regression, with only 2 parameters, and knowing that the parameters are the intercept and slope of the line, sklearn can estimate them directly from our data. Notice that all of the data must be available to traverse and calculate the parameters.

Plot outputs

we can plot the fit line over the data:

```
In [12]: plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color='blue')
plt.plot(train_x, regr.coef_[0]*train_x + regr.intercept_[0], '-r')
plt.xlabel("Engine size")
plt.ylabel("Emission")
```

```
Out[12]: Text(0, 0.5, 'Emission')
```



Evaluation

we compare the actual values and predicted values to calculate the accuracy of a regression model. Evaluation metrics provide a key role in the development of a model, as it provides insight to areas that require improvement.

There are different model evaluation metrics, lets use MSE here to calculate the accuracy of our model based on the test set:

- Mean absolute error: It is the mean of the absolute value of the errors. This is the easiest of the metrics to understand since it's just average error.
- Mean Squared Error (MSE): Mean Squared Error (MSE) is the mean of the squared error. It's more popular than Mean absolute error because the focus is geared more towards large errors. This is due to the squared term exponentially increasing larger errors in comparison to smaller ones.
- Root Mean Squared Error (RMSE).
- R-squared is not error, but is a popular metric for accuracy of your model. It represents how close the data are to the fitted regression line. The higher the R-squared, the better the model fits your data. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse).

```
In [13]: from sklearn.metrics import r2_score

test_x = np.asarray(test[['ENGINESIZE']])
test_y = np.asarray(test[['CO2EMISSIONS']])
test_y_ = regr.predict(test_x)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y_ , test_y))
```

Mean absolute error: 25.30
 Residual sum of squares (MSE): 1144.07
 R2-score: 0.62

In []:

Часть 3

Multiple Linear Regression

</h1>

Теоретические сведения

In this notebook, we learn how to use scikit-learn to implement Multiple linear regression. We download a dataset that is related to fuel consumption and Carbon dioxide emission of cars. Then, we split our data into training and test sets, create a model using training set, Evaluate your model using test set, and finally use model to predict unknown value

В этой практической работе мы узнаем, как использовать scikit-learn для реализации множественной линейной регрессии. Мы загружаем набор данных, который связан с расходом топлива и выбросами углекислого газа автомобилей. Затем мы разделяем наши данные на обучающие и тестовые наборы, создаем модель с использованием обучающего набора, оцениваем вашу модель с помощью тестового набора и, наконец, используем модель для прогнозирования неизвестного значения.

Программа работы

1. Understanding the Data
2. Reading the Data in
3. Multiple Regression Model
4. Prediction
5. Practice

Importing Needed packages

```
In [1]: import matplotlib.pyplot as plt  
import pandas as pd  
import pylab as pl  
import numpy as np  
%matplotlib inline
```

Understanding the Data

FuelConsumption.csv :

We have downloaded a fuel consumption dataset, **FuelConsumption.csv**, which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada. [Dataset source](#)

- **MODELYEAR** e.g. 2014
- **MAKE** e.g. Acura
- **MODEL** e.g. ILX
- **VEHICLE CLASS** e.g. SUV
- **ENGINE SIZE** e.g. 4.7
- **CYLINDERS** e.g 6
- **TRANSMISSION** e.g. A6
- **FUELTYPE** e.g. Z
- **FUEL CONSUMPTION in CITY(L/100 km)** e.g. 9.9
- **FUEL CONSUMPTION in HWY (L/100 km)** e.g. 8.9
- **FUEL CONSUMPTION COMB (L/100 km)** e.g. 9.2
- **CO2 EMISSIONS (g/km)** e.g. 182 --> low --> 0

Reading the data in

```
In [2]: df = pd.read_csv("FuelConsumption.csv")  
  
# take a look at the dataset  
df.head()
```

```
Out[2]:
```

	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION	FUELTYPE	FL
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5	Z	
1	2014	ACURA	ILX	COMPACT	2.4	4	M6	Z	
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7	Z	
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6	Z	
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6	Z	

Lets select some features that we want to use for regression.

```
In [3]: cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_CITY','FUELCONSUMPTION_HWY','FUELCON  
cdf.head(9)
```

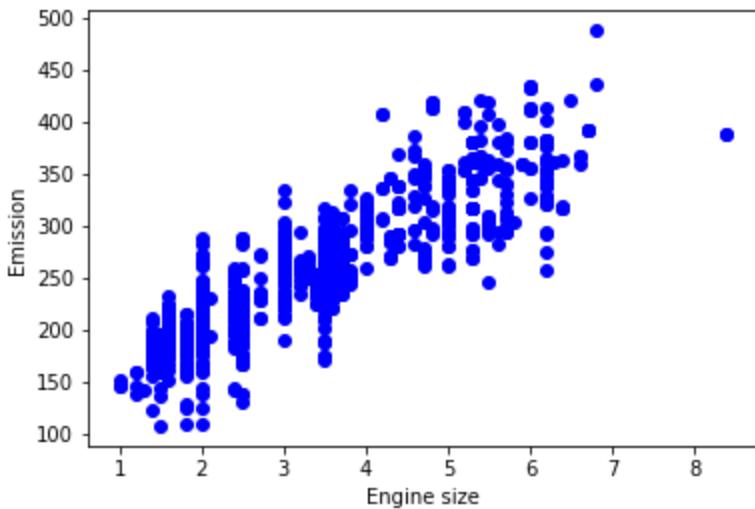
```
Out[3]:
```

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_CITY	FUELCONSUMPTION_HWY	FUELCONSUMPTION_CO
0	2.0	4	9.9	6.7	
1	2.4	4	11.2	7.7	
2	1.5	4	6.0	5.8	
3	3.5	6	12.7	9.1	
4	3.5	6	12.1	8.7	1
5	3.5	6	11.9	7.7	1
6	3.5	6	11.8	8.1	

7	3.7	6	12.8	9.0
8	3.7	6	13.4	9.5

Lets plot Emission values with respect to Engine size:

```
In [4]: plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS, color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



Creating train and test dataset

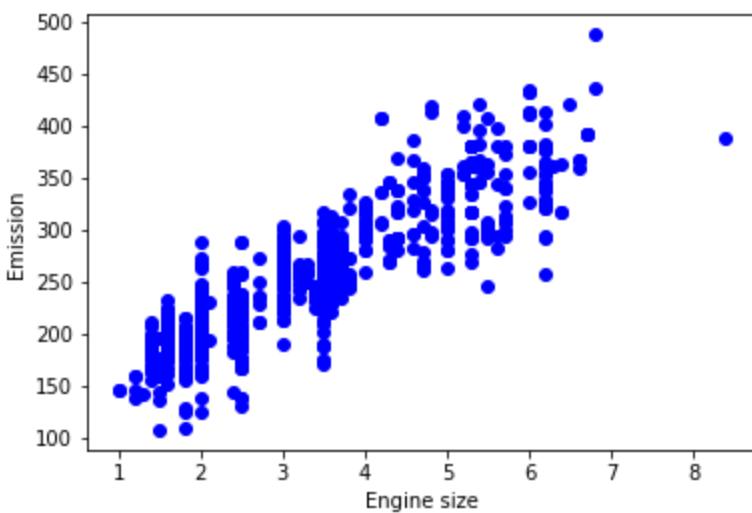
Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the data. It is more realistic for real world problems.

This means that we know the outcome of each data point in this dataset, making it great to test with! And since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. So, in essence, it's truly an out-of-sample testing.

```
In [5]: msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

Train data distribution

```
In [6]: plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS, color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```



Multiple Regression Model

In reality, there are multiple variables that predict the Co2emission. When more than one independent variable is present, the process is called multiple linear regression. For example, predicting co2emission using FUELCONSUMPTION_COMB, EngineSize and Cylinders of cars. The good thing here is that Multiple linear regression is the extension of simple linear regression model.

```
In [7]: from sklearn import linear_model
regr = linear_model.LinearRegression()
x = np.asarray(train[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']])
y = np.asarray(train[['CO2EMISSIONS']])
regr.fit (x, y)
# The coefficients
print ('Coefficients: ', regr.coef_)

Coefficients:  [[10.74744339  8.14011667  9.35137832]]
```

As mentioned before, **Coefficient** and **Intercept**, are the parameters of the fit line. Given that it is a multiple linear regression, with 3 parameters, and knowing that the parameters are the intercept and coefficients of hyperplane, sklearn can estimate them from our data. Scikit-learn uses plain Ordinary Least Squares method to solve this problem.

Ordinary Least Squares (OLS)

OLS is a method for estimating the unknown parameters in a linear regression model. OLS chooses the parameters of a linear function of a set of explanatory variables by minimizing the sum of the squares of the differences between the target dependent variable and those predicted by the linear function. In other words, it tries to minimizes the sum of squared errors (SSE) or mean squared error (MSE) between the target variable (y) and our predicted output (\hat{y}) over all samples in the dataset.

OLS can find the best parameters using of the following methods:

- Solving the model parameters analytically using closed-form equations
- Using an optimization algorithm (Gradient Descent, Stochastic Gradient Descent, Newton's Method, etc.)

Prediction

```
In [8]: y_hat= regr.predict(test[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']])
```

```

x = np.asarray(test[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']])
y = np.asarray(test[['CO2EMISSIONS']])
print("Residual sum of squares: %.2f"
      % np.mean((y_hat - y) ** 2))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regr.score(x, y))

```

Residual sum of squares: 608.81
 Variance score: 0.84

explained variance regression score:

If \hat{y} is the estimated target output, y the corresponding (correct) target output, and Var is Variance, the square of the standard deviation, then the explained variance is estimated as follow:

$$\text{explainedVariance}(y, \hat{y}) = 1 - \frac{\text{Var}\{y - \hat{y}\}}{\text{Var}\{y\}}$$

The best possible score is 1.0, lower values are worse.

Practice

Try to use a multiple linear regression with the same dataset but this time use **FUEL CONSUMPTION in CITY** and **FUEL CONSUMPTION in HWY** instead of FUELCONSUMPTION_COMB. Does it result in better accuracy?

In [9]: # write your code here

Double-click **here** for the solution.

Часть 4

Non Linear Regression Analysis

</h1>

Теоретические сведения

If the data shows a curvy trend, then linear regression will not produce very accurate results when compared to a non-linear regression because, as the name implies, linear regression presumes that the data is linear. Let's learn about non linear regressions and apply an example on python. In this notebook, we fit a non-linear model to the datapoints corresponding to China's GDP from 1960 to 2014.

Если данные показывают кривую тенденцию, то линейная регрессия не даст очень точных результатов по сравнению с нелинейной регрессией, потому что, как следует из названия, линейная регрессия предполагает, что данные линейны.

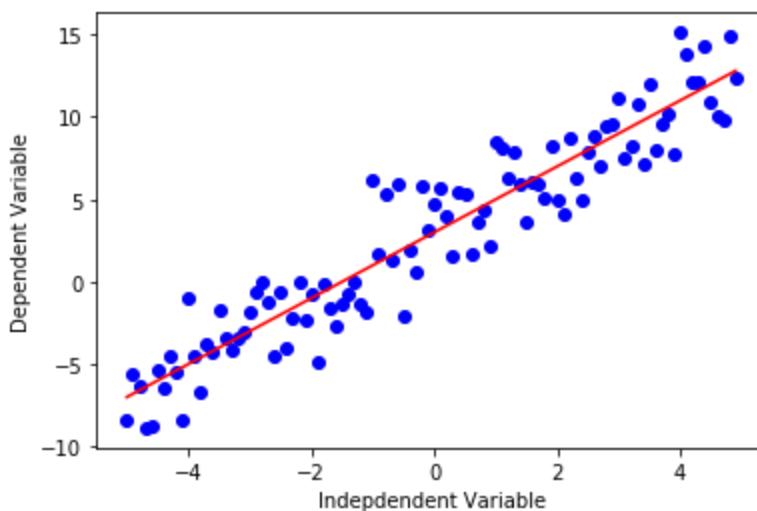
В данной практической работе мы используем нелинейную регрессию на примере. В частности, мы подгоняем нелинейную модель к точкам данных, соответствующим ВВП Китая с 1960 по 2014 год.

Importing required libraries

```
In [1]: import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline
```

Though Linear regression is very good to solve many problems, it cannot be used for all datasets. First recall how linear regression could model a dataset. It models a linear relation between a dependent variable y and independent variable x . It had a simple equation, of degree 1, for example $y = 2x + 3$.

```
In [2]: x = np.arange(-5.0, 5.0, 0.1)  
  
##You can adjust the slope and intercept to verify the changes in the graph  
y = 2*(x) + 3  
y_noise = 2 * np.random.normal(size=x.size)  
ydata = y + y_noise  
plt.figure(figsize=(8,6))  
plt.plot(x, ydata, 'bo')  
plt.plot(x,y, 'r')  
plt.ylabel('Dependent Variable')  
plt.xlabel('Independent Variable')  
plt.show()
```



Non-linear regressions are a relationship between independent variables x and a dependent variable y which result in a non-linear function modeled data. Essentially any relationship that is not linear can be termed as non-linear, and is usually represented by the polynomial of k degrees (maximum power of x).

$$y = ax^3 + bx^2 + cx + d$$

Non-linear functions can have elements like exponentials, logarithms, fractions, and others. For example:

$$y = \log(x)$$

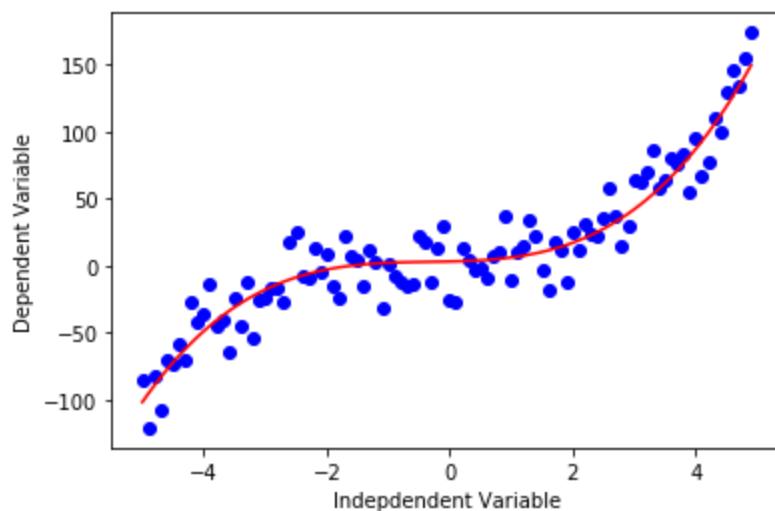
Or even, more complicated such as :

$$y = \log(ax^3 + bx^2 + cx + d)$$

Let's take a look at a cubic function's graph.

```
In [3]: x = np.arange(-5.0, 5.0, 0.1)

## You can adjust the slope and intercept to verify the changes in the graph
y = 1*(x**3) + 1*(x**2) + 1*x + 3
y_noise = 20 * np.random.normal(size=x.size)
ydata = y + y_noise
plt.plot(x, ydata, 'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
plt.xlabel('Independent Variable')
plt.show()
```



As you can see, this function has x^3 and x^2 as independent variables. Also, the graphic of this function is not a straight line over the 2D plane. So this is a non-linear function.

Some other types of non-linear functions are:

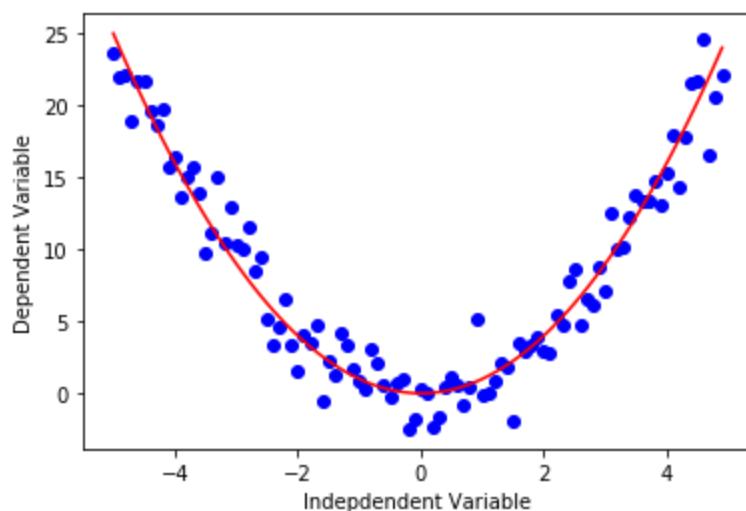
Quadratic

$$Y = X^2$$

```
In [4]: x = np.arange(-5.0, 5.0, 0.1)

##You can adjust the slope and intercept to verify the changes in the graph

y = np.power(x,2)
y_noise = 2 * np.random.normal(size=x.size)
ydata = y + y_noise
plt.plot(x, ydata, 'bo')
plt.plot(x,y, 'r')
plt.ylabel('Dependent Variable')
plt.xlabel('Independent Variable')
plt.show()
```



Exponential

An exponential function with base c is defined by

$$Y = a + bc^X$$

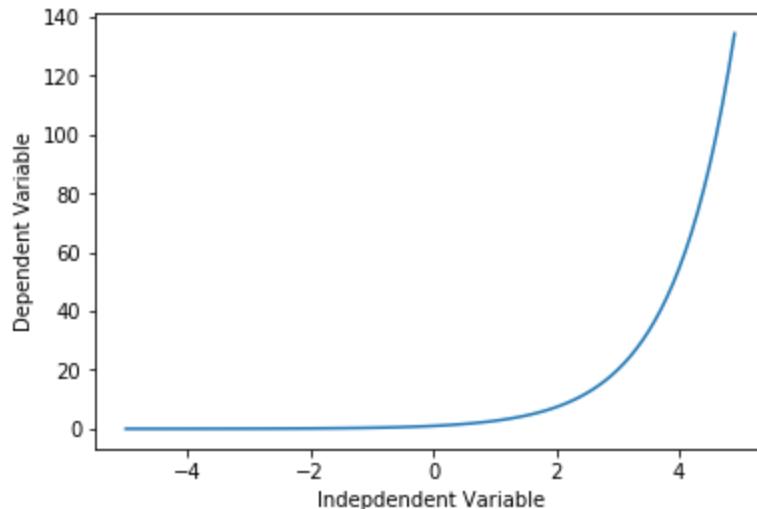
where $b \neq 0$, $c > 0$, $c \neq 1$, and x is any real number. The base, c , is constant and the exponent, x , is a variable.

```
In [5]: X = np.arange(-5.0, 5.0, 0.1)

##You can adjust the slope and intercept to verify the changes in the graph

Y= np.exp(X)

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Independent Variable')
plt.show()
```



Logarithmic

The response y is a results of applying logarithmic map from input x 's to output variable y . It is one of the simplest form of **log()**: i.e.

$$y = \log(x)$$

Please consider that instead of x , we can use X , which can be polynomial representation of the x 's. In general form it would be written as

$$y = \log(X) \tag{1}$$

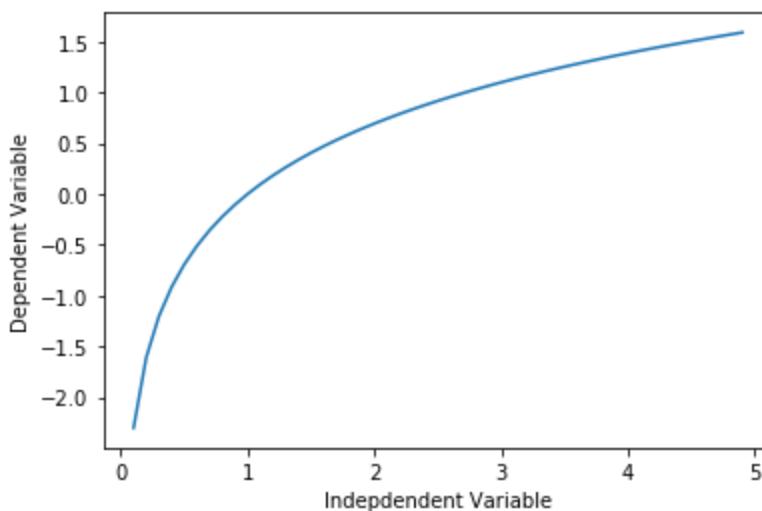
```
In [6]: X = np.arange(-5.0, 5.0, 0.1)

Y = np.log(X)

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Independent Variable')
plt.show()
```

```
J:\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: RuntimeWarning: invalid value encountered in log
```

```
This is separate from the ipykernel package so we can avoid doing imports until
```



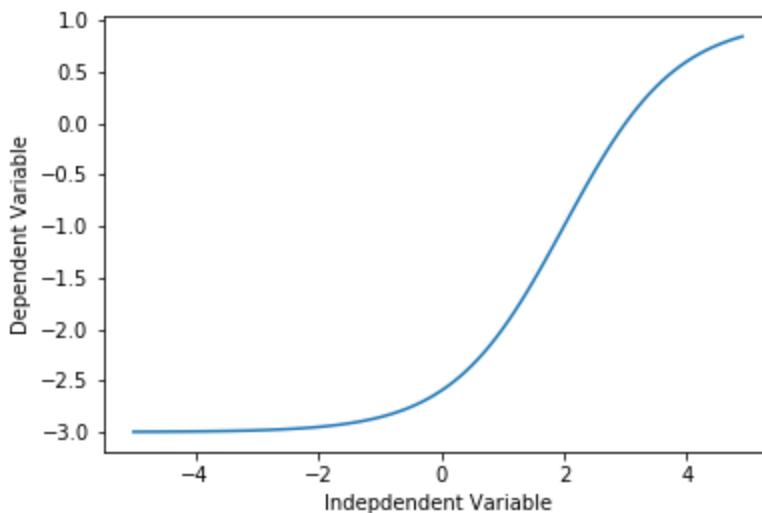
Sigmoidal/Logistic

$$Y = a + \frac{b}{1 + c^{(X-d)}}$$

```
In [7]: X = np.arange(-5.0, 5.0, 0.1)

Y = 1-4/(1+np.power(3, X-2))

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Independent Variable')
plt.show()
```



Non-Linear Regression example

For an example, we're going to try and fit a non-linear model to the datapoints corresponding to China's GDP from 1960 to 2014. We download a dataset with two columns, the first, a year between 1960 and 2014, the second, China's corresponding annual gross domestic income in US dollars for that year.

```
In [8]: import numpy as np
import pandas as pd
```

```
#downloading dataset
#!wget -nv -O china_gdp.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses
```

```
df = pd.read_csv("china_gdp.csv")
df.head(10)
```

Out [8]:

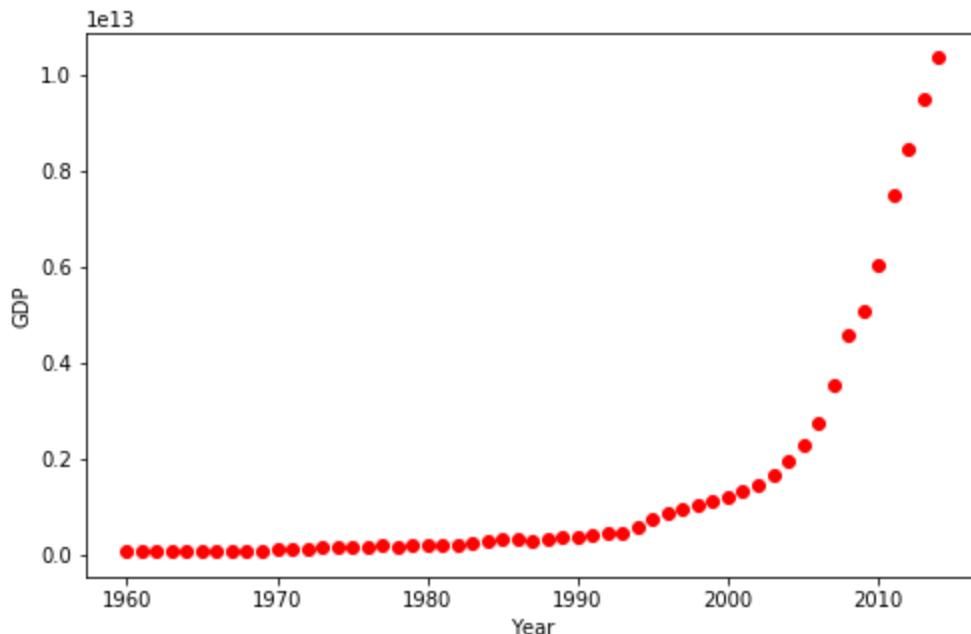
	Year	Value
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10
5	1965	6.970915e+10
6	1966	7.587943e+10
7	1967	7.205703e+10
8	1968	6.999350e+10
9	1969	7.871882e+10

Plotting the Dataset

This is what the datapoints look like. It kind of looks like an either logistic or exponential function. The growth starts off slow, then from 2005 on forward, the growth is very significant. And finally, it decelerate slightly in the 2010s.

In [9]:

```
plt.figure(figsize=(8,5))
x_data, y_data = (df["Year"].values, df["Value"].values)
plt.plot(x_data, y_data, 'ro')
plt.ylabel('GDP')
plt.xlabel('Year')
plt.show()
```

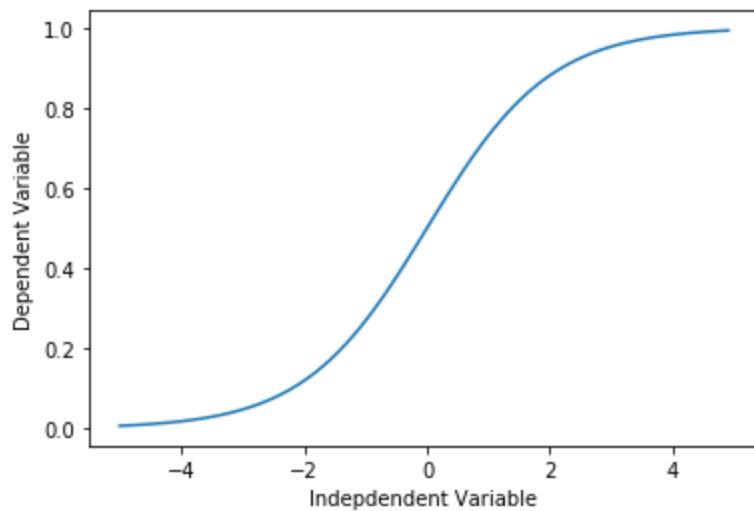


Choosing a model

From an initial look at the plot, we determine that the logistic function could be a good approximation, since it has the property of starting with a slow growth, increasing growth in the middle, and then decreasing again at the end; as illustrated below:

```
In [10]: X = np.arange(-5.0, 5.0, 0.1)
Y = 1.0 / (1.0 + np.exp(-X))

plt.plot(X,Y)
plt.ylabel('Dependent Variable')
plt.xlabel('Independent Variable')
plt.show()
```



The formula for the logistic function is the following:

$$\hat{Y} = \frac{1}{1 + e^{\beta_1(X - \beta_2)}}$$

β_1 : Controls the curve's steepness,

β_2 : Slides the curve on the x-axis.

Building The Model

Now, let's build our regression model and initialize its parameters.

```
In [11]: def sigmoid(x, Beta_1, Beta_2):
    y = 1 / (1 + np.exp(-Beta_1*(x-Beta_2)))
    return y
```

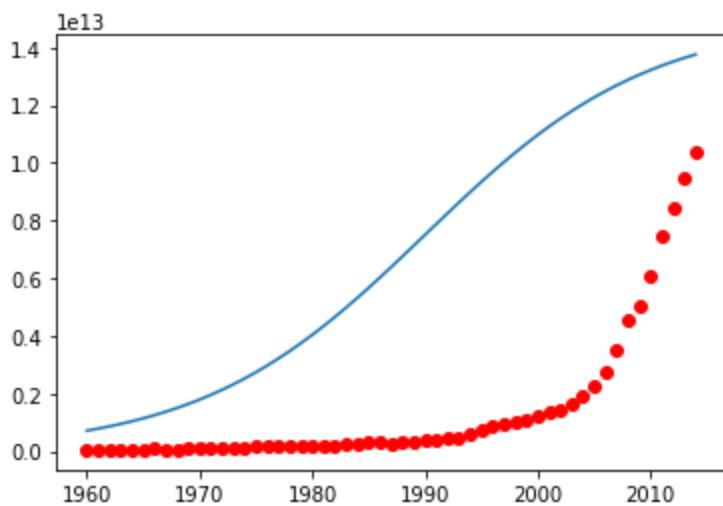
Lets look at a sample sigmoid line that might fit with the data:

```
In [12]: beta_1 = 0.10
beta_2 = 1990.0

#logistic function
Y_pred = sigmoid(x_data, beta_1 , beta_2)

#plot initial prediction against datapoints
plt.plot(x_data, Y_pred*1500000000000.)
plt.plot(x_data, y_data, 'ro')
```

```
Out[12]: []
```



Our task here is to find the best parameters for our model. Lets first normalize our x and y:

```
In [13]: # Lets normalize our data
xdata = x_data/max(x_data)
ydata = y_data/max(y_data)
```

How we find the best parameters for our fit line?

we can use **curve_fit** which uses non-linear least squares to fit our sigmoid function, to data. Optimal values for the parameters so that the sum of the squared residuals of `sigmoid(xdata, *popt) - ydata` is minimized.

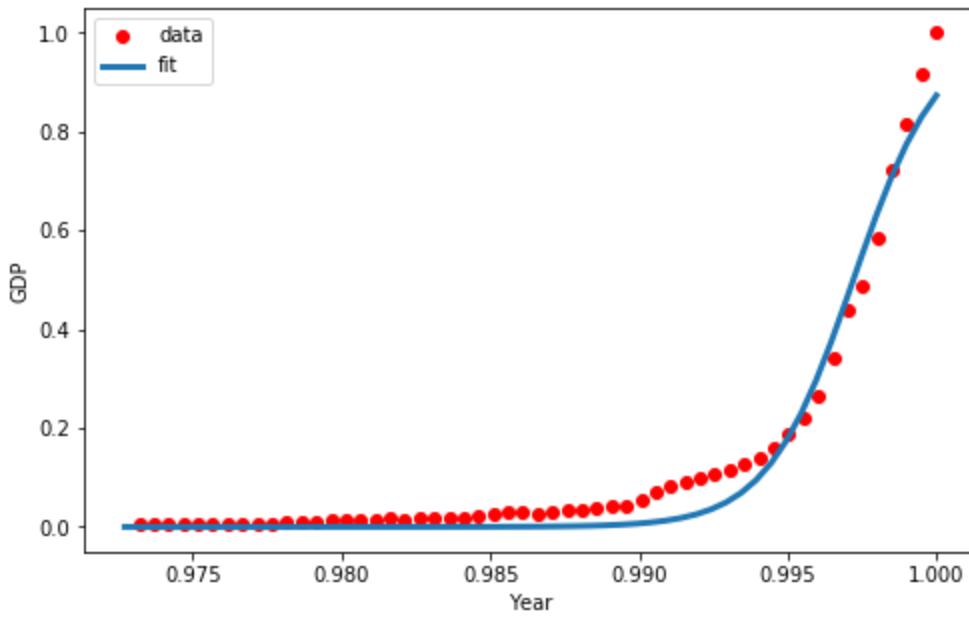
`popt` are our optimized parameters.

```
In [14]: from scipy.optimize import curve_fit
popt, pcov = curve_fit(sigmoid, xdata, ydata)
#print the final parameters
print(" beta_1 = %f, beta_2 = %f" % (popt[0], popt[1]))
```

beta_1 = 690.451711, beta_2 = 0.997207

Now we plot our resulting regression model.

```
In [15]: x = np.linspace(1960, 2015, 55)
x = x/max(x)
plt.figure(figsize=(8,5))
y = sigmoid(x, *popt)
plt.plot(xdata, ydata, 'ro', label='data')
plt.plot(x,y, linewidth=3.0, label='fit')
plt.legend(loc='best')
plt.ylabel('GDP')
plt.xlabel('Year')
plt.show()
```



Practice

Can you calculate what is the accuracy of our model?

In [16]: `# write your code here`

Double-click **here** for the solution.

Практическая работа №3
Визуализация данных средствами Matplotlib

Цель работы: получить навыки использования библиотеки визуализации данных Matplotlib с использованием языка программирования Python.

Часть 1

Основы

matplotlib – это основная библиотека для построения научных графиков в Python. Она включает функции для создания высококачественных визуализаций типа линейных диаграмм, гистограмм, диаграмм разброса и т.д. Визуализация данных и различных аспектов вашего анализа может дать важную информацию.

В данной работе взаимодействие с matplotlib будет проходить в Jupyter Notebook (см. Методические указания к Практическому занятию №2.1) на базе Google Colab (см. <https://colab.research.google.com/notebooks/intro.ipynb>, <https://github.com/deepmipt/dlschl/wiki/Инструкция-по-работе-с-Google-Colab>).

В среде Jupyter Notebook возможно вывести рисунок прямо в браузере с помощью встроенных команд `%matplotlib notebook` и `%matplotlib inline`. Рекомендуется использовать `%matplotlib inline`.

Использование Google Colab позволяет не устанавливать на свой компьютер Jupyter Notebook.

1. Подготовительная часть.

1.1. Зарегистрировать электронную почту google (либо использовать существующий аккаунт).

1.2. Перейти по ссылке <https://colab.research.google.com/notebooks/intro.ipynb>

1.3. В правом верхнем углу нажать кнопку «Войти» и затем ввести свои учетные данные google.

1.4. В верхнем левом углу найдите подменю «Файл», далее «Создать блокнот».

2. Опробовать программу для построения 2D графиков со следующим текстом.

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
# Генерируем последовательность чисел от -10 до 10 с 100 шагами
x = np.linspace(-10, 10, 100)
# Генерируем случайную амплитуду для синусоиды
```

```
a = np.random.random()
# Создаем второй массив с помощью синуса
y = a*np.sin(x)
# Функция создает линейный график на основе двух массивов
plt.plot(x, y, marker="x")
```

Пример результата показан на рисунке.

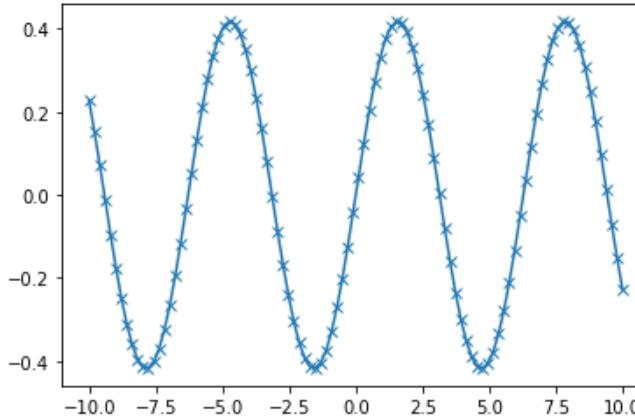


Рис. Результат работы программы

3. Работа с данными, загруженными из открытых источников сети интернет.

3.0. В рамках данного пункта лабораторной работы будут использованы библиотеки Python pandas (<https://pandas.pydata.org/>), Numpy (<https://numpy.org/>). Стоит отметить, что библиотека pandas имеет встроенный построитель графиков plot, который и будет использоваться в данном пункте. Будет использован набор данных (dataset) об Иммиграции в Канаду с 1980 по 2013 год - Международная миграция в отдельные страны и из них - Редакция 2015 года с веб-сайта Организации Объединенных Наций (<https://www.un.org/en/development/desa/population/migration/data/empirical2/migrationflows.shtml>). Набор данных содержит годовые данные о потоках международных мигрантов, регистрируемых различными странами. Данные показывают как приток, так и отток в зависимости от места рождения, гражданства или места предыдущего / следующего проживания как для иностранцев, так и для граждан. В рамках данного

пункта мы сосредоточимся на данных иммиграционной службы Канады.

3.1. Загрузка и подготовка данных.

3.1.1. Импорт первичных библиотек - pandas, Numpy.

```
import numpy as np  
import pandas as pd
```

3.1.2. Загрузка данных из сети интернет в pandas *dataframe*.

```
df_can = pd.read_excel('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DV0101EN/labs/Data_Files/Canada.xlsx',  
                      sheet_name='Canada by Citizenship',  
                      skiprows=range(20),  
                      skipfooter=2  
)  
print('Данные загружены и записаны в dataframe!')
```

3.1.3. Обзор данных – первые 5 элементов:

```
df_can.head()
```

3.1.4. Обзор данных – размер (строки и столбы) dataset'a:

```
print(df_can.shape)
```

3.1.5. Очистка данных – удаление неинформативных для нас столбцов, повторный вывод первых 5 строк:

```
df_can.drop(['AREA', 'REG', 'DEV', 'Type', 'Coverage'], axis=1, inplace=True)  
df_can.head()
```

3.1.6. Приведение данных к более удобному виду – переименование нескольких столбцов, повторный вывод первых 5 строк:

```
df_can.rename(columns={'OdName':'Country', 'AreaName':'Continent','RegName':'Region'}, inplace=True)  
df_can.head()
```

3.1.7. Проверка структуры данных – уточняем, являются ли наименования всех столбцов типами «строка» («string»):

```
all(isinstance(column, str) for column in df_can.columns)
```

Результатом будет скорее всего False. Поэтому выполняем преобразование.

3.1.8. Изменяем наименование всех столбцов так, чтобы они были типа string и проверяем заново:

```
df_can.columns = list(map(str, df_can.columns))  
all(isinstance(column, str) for column in df_can.columns)
```

3.1.9. Приведение данных к более удобному виду – задаем в качестве строчного индекса наименование страны, повторный вывод первых 5 строк:

```
df_can.set_index('Country', inplace=True)
df_can.head()
```

3.1.10. Расширяем данные – создаем новый столбец Total, который будет являться суммой всех столбцов (фактически – количеством иммигрантов за все годы с 1980 по 2013), повторный вывод первых 5 строк:

```
df_can['Total'] = df_can.sum(axis=1)
df_can.head()
```

3.1.11. Создаем новый набор данных на базе предыдущего – выделяем в него 5 стран, иммиграция из которых больше всех остальных:

```
years = list(map(str, range(1980, 2014)))
df_can.sort_values(['Total'], ascending=False, axis=0, inplace=True)
df_top5 = df_can.head()
# Транспонирование таблицы
df_top5 = df_top5[years].transpose()
df_top5.head()
```

3.2. Вывод данных в виде графика типа «Диаграмма с областями»:

```
%matplotlib inline
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.style.use('ggplot') # опционально: задаем стиль ggplot

# Проверяем версию Matplotlib
print ('Matplotlib version: ', mpl.__version__) # >= 2.0.0

# Для построения графика изменяем тип индексов строк (годы)
# на integer
df_top5.index = df_top5.index.map(int)
# Построение графика типа 'area' встроенной
# в pandas суб-библиотекой matplotlib
df_top5.plot(kind='area',
```

```
stacked=False,  
figsize=(20, 10), # размер области построения графика  
)  
#Задаем наименование графика  
plt.title('Immigration Trend of Top 5 Countries')  
#Задаем наименование оси Y  
plt.ylabel('Number of Immigrants')  
#Задаем наименование оси X  
plt.xlabel('Years')  
# Выводим график со всеми параметрами на экран  
plt.show()
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличается построение графиков с помощью matplotlib и pandas?
2. Какое значение параметра kind нужно задать функции plot для вывода графика типа «Диаграмма с областями»?

Часть 2

Диаграммы

matplotlib – это основная библиотека для построения научных графиков в Python. Она включает функции для создания высококачественных визуализаций типа линейных диаграмм, гистограмм, диаграмм разброса и т.д. Визуализация данных и различных аспектов вашего анализа может дать важную информацию.

В данной работе взаимодействие с matplotlib будет проходить в Jupyter Notebook на базе Google Colab (см. <https://colab.research.google.com/notebooks/intro.ipynb>, [https://github.com/deepmipt/dlschl/wiki/Инструкция-по-工作中-с-Google-Colab](https://github.com/deepmipt/dlschl/wiki/Инструкция-по-работе-с-Google-Colab)).

В среде Jupyter Notebook возможно вывести рисунок прямо в браузере с помощью встроенных команд %matplotlib notebook и %matplotlib inline. Рекомендуется использовать %matplotlib inline.

Использование Google Colab позволяет не устанавливать на свой компьютер Jupyter Notebook.

1. Подготовительная часть.

1.1. Зарегистрировать электронную почту google (либо использовать существующий аккаунт).

1.2. Перейти по ссылке <https://colab.research.google.com/notebooks/intro.ipynb>

1.3. В правом верхнем углу нажать кнопку «Войти» и затем ввести свои учетные данные google.

1.4. В верхнем левом углу найдите подменю «Файл», далее «Открыть блокнот» и выберите блокнот, который вы создали в рамках ч. 1.

1.5. Выполните заново все ячейки этого блокнота.

2. Построение Гистограмм.

Гистограмма - это способ представления частотного распределения числового набора данных. Она работает так: ось x разбивается на ячейки, каждая точка данных в наборе данных назначается ячейке, а затем подсчитывается количество точек данных, назначенных каждой ячейке. Таким образом, ось Y - это частота или количество точек данных в каждой ячейке. Обратите внимание, что мы можем изменять размеры, и обычно их нужно настроить, чтобы распределение отображалось красиво.

2.1. Обзор данных

```
df_can['2013'].head()
```

```
df_can['2013'].head()
```

Country	2013
India	33087
China	34129
United Kingdom of Great Britain and Northern Ireland	5827
Philippines	29544
Pakistan	12603

Name: 2013, dtype: int64

2.2. Подготовка данных для гистограммы.

```
# np.histogram возвращает два значения
count, bin_edges = np.histogram(df_can['2013'])
```

```
print(count) # подсчет частоты появления данных
print(bin_edges) # количество столбцов, по умолчанию – 10
```

```
# np.histogram returns 2 values
count, bin_edges = np.histogram(df_can['2013'])

print(count) # frequency count
print(bin_edges) # bin ranges, default = 10 bins
```

```
[178 11 1 2 0 0 0 0 1 2]
[ 0. 3412.9 6825.8 10238.7 13651.6 17064.5 20477.4 23890.3 27303.2
 30716.1 34129. ]
```

2.3. Построение гистограммы:

```
df_can['2013'].plot(kind='hist', figsize=(8, 5))
```

```
plt.title('Histogram of Immigration from 195 Countries in 2013') # добавление названия
plt.ylabel('Number of Countries') # добавление наименования оси y
plt.xlabel('Number of Immigrants') # наименование оси x
```

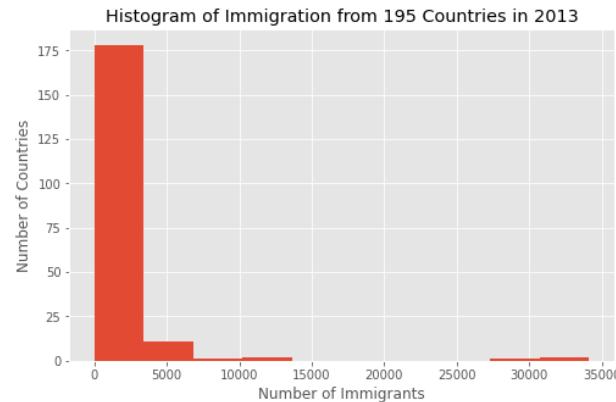
```

plt.show()
df_can['2013'].plot(kind='hist', figsize=(8, 5))

plt.title('Histogram of Immigration from 195 Countries in 2013') # add a title to the histogram
plt.ylabel('Number of Countries') # add y-label
plt.xlabel('Number of Immigrants') # add x-label

plt.show()

```



3. Построение Bar Charts (Dataframe):

Bar Charts – это способ представления данных, где длина полосок представляет величину / размер функции / переменной. Bar Charts обычно представляют числовые и категориальные переменные, сгруппированные по интервалам.

3.1. Извлекаем часть данных из df_can:

```

# step 1: get the data
df_iceland = df_can.loc['Iceland', years]
df_iceland.head()

# step 1: get the data
df_iceland = df_can.loc['Iceland', years]

```

```
df_iceland.head()
```

1980	17
1981	33
1982	10
1983	9
1984	13

```
Name: Iceland, dtype: object
```

3.2. Построение графика (горизонтальный Bar Chart):

```
# step 2: plot data
```

```
df_iceland.plot(kind='barh', figsize=(10, 6))
```

```
plt.xlabel('Year') # add to x-label to the plot
```

```
plt.ylabel('Number of immigrants') # add y-label to the plot
```

```
plt.title('Icelandic immigrants to Canada from 1980 to 2013') # add title to the plot
```

```
plt.show()
```

```
# step 2: plot data
```

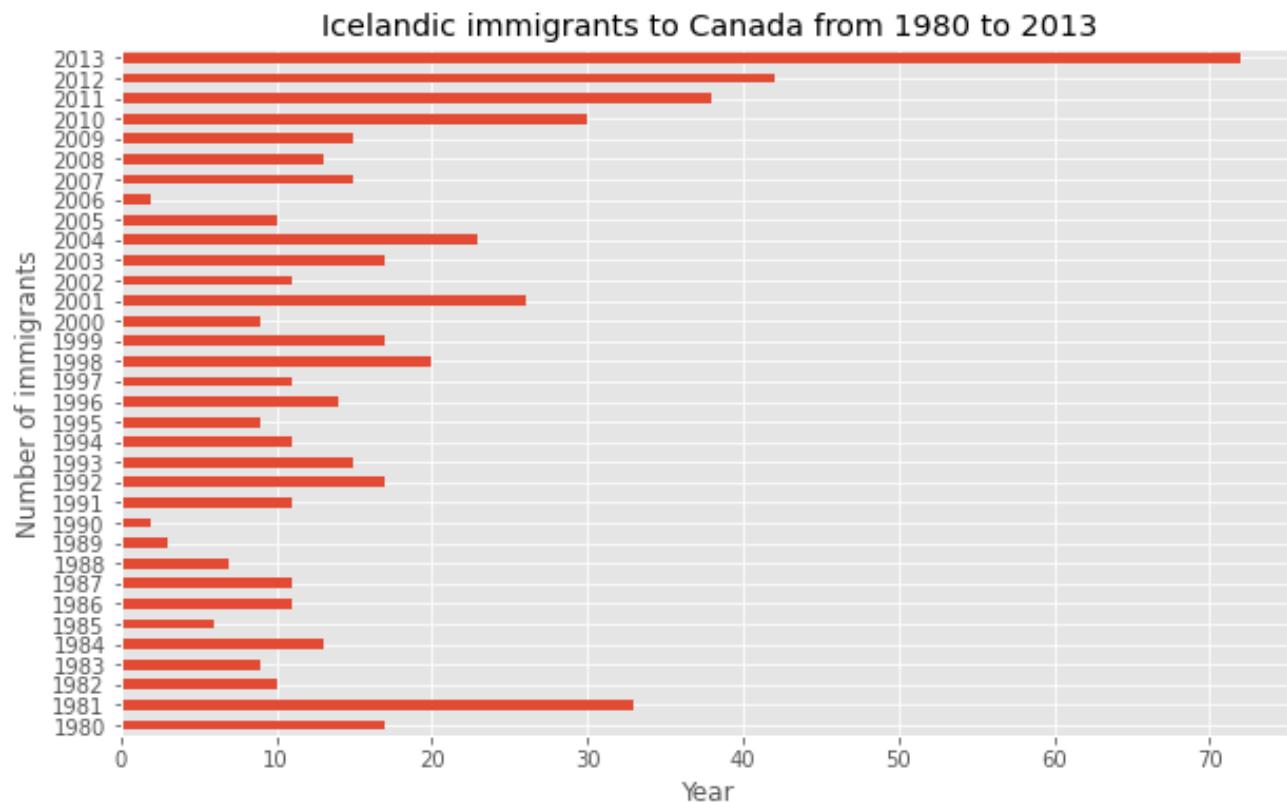
```
df_iceland.plot(kind='barh', figsize=(10, 6))
```

```
plt.xlabel('Year') # add to x-label to the plot
```

```
plt.ylabel('Number of immigrants') # add y-label to the plot
```

```
plt.title('Icelandic immigrants to Canada from 1980 to 2013') # add title to the plot
```

```
plt.show()
```



КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В п. 3.2 попробуйте изменить `kind='barh'` на `kind='bar'`, что получится?

Часть 3

K-Means Clustering

Теоретические сведения

There are many models for **clustering** out there. In this notebook, we will be presenting the model that is considered one of the simplest models amongst them. Despite its simplicity, the **K-means** is vastly used for clustering in many data science applications, especially useful if you need to quickly discover insights from **unlabeled data**. In this notebook, you will learn how to use k-Means for customer segmentation.

Some real-world applications of k-means:

- Customer segmentation
- Understand what the visitors of a website are trying to accomplish
- Pattern recognition
- Machine learning
- Data compression

In this notebook we practice k-means clustering with 2 examples:

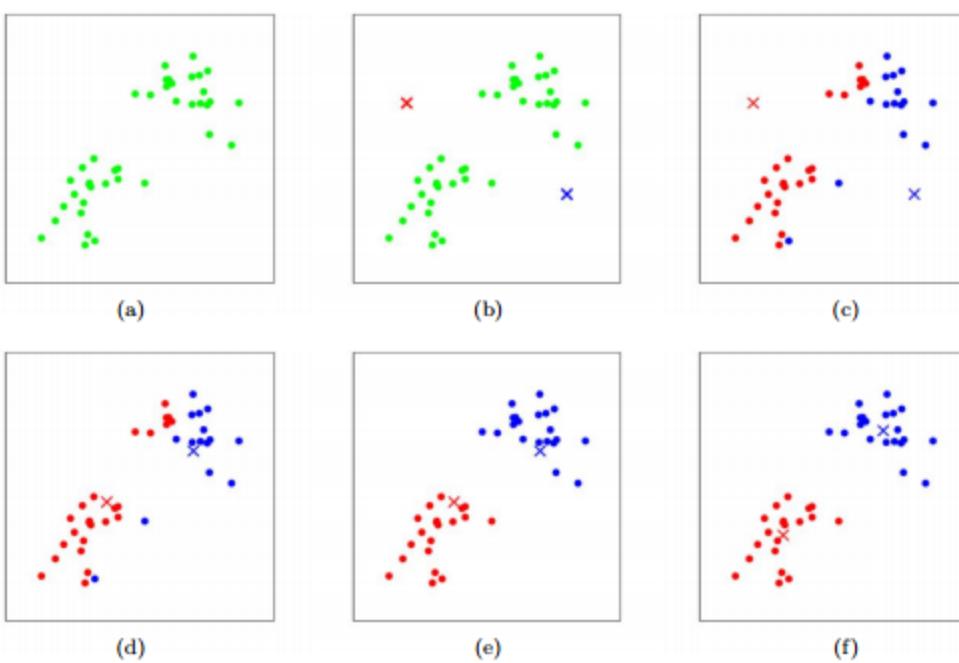
- k-means on a random generated dataset
- Using k-means for customer segmentation

Существует множество моделей кластеризации. В этом лабораторной работе мы представим модель, которая считается одной из самых простых среди них. Несмотря на свою простоту, метод К-средних широко используется для кластеризации во многих приложениях по обработке и анализу данных, что особенно полезно, если вам нужно быстро извлечь ценную информацию из неразмеченных данных. В этой лабораторной работе вы узнаете, как использовать k-Means для сегментации клиентов.

Действие алгоритма таково, что он стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров. Он разбивает множество элементов векторного пространства на заранее известное число кластеров k .

Основная идея заключается в том, что на каждой итерации пересчитываются центр масс 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.



Некоторые реальные приложения k-средних:

- Сегментация клиентов
- Распознавание образов
- Машинное обучение
- Сжатие данных

В этой лабораторной работе мы используем кластеризацию методом k-средних на двух примерах:

k-средних для случайно сгенерированного набора данных
Использование k-средних для сегментации клиентов

Программа работы

- k-Means on a randomly generated dataset
 1. Setting up K-Means
 2. Creating the Visual Plot
- Customer Segmentation with K-Means
 1. Pre-processing
 2. Modeling
 3. Insights

Import libraries

Lets first import the required libraries. Also run `%matplotlib inline` since we will be plotting in this section.

```
In [1]: import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
```

k-Means on a randomly generated dataset

Lets create our own dataset for this lab!

First we need to set up a random seed. Use **numpy's random.seed()** function, where the seed will be set to **0**

```
In [2]: np.random.seed(0)
```

Next we will be making *random clusters* of points by using the **make_blobs** class. The **make_blobs** class can take in many inputs, but we will be using these specific ones.

Input

- **n_samples:** The total number of points equally divided among clusters.
 - Value will be: 5000
- **centers:** The number of centers to generate, or the fixed center locations.
 - Value will be: [[4, 4], [-2, -1], [2, -3],[1,1]]
- **cluster_std:** The standard deviation of the clusters.
 - Value will be: 0.9

Output

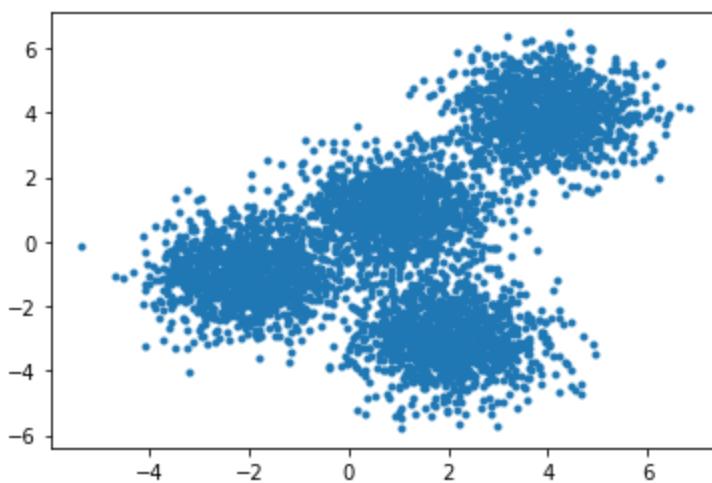
- **X:** Array of shape [n_samples, n_features]. (Feature Matrix)
 - The generated samples.
- **y:** Array of shape [n_samples]. (Response Vector)
 - The integer labels for cluster membership of each sample.

```
In [3]: X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster_st
```

Display the scatter plot of the randomly generated data.

```
In [4]: plt.scatter(X[:, 0], X[:, 1], marker='.'
```

```
Out[4]: <matplotlib.collections.PathCollection at 0x2c062270148>
```



Setting up K-Means

Now that we have our random data, let's set up our K-Means Clustering.

The KMeans class has many parameters that can be used, but we will be using these three:

- **init**: Initialization method of the centroids.
 - Value will be: "k-means++"
 - k-means++: Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
- **n_clusters**: The number of clusters to form as well as the number of centroids to generate.
 - Value will be: 4 (since we have 4 centers)
- **n_init**: Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
 - Value will be: 12

Initialize KMeans with these parameters, where the output parameter is called **k_means**.

```
In [5]: k_means = KMeans(init = "k-means++", n_clusters = 4, n_init = 12)
```

Now let's fit the KMeans model with the feature matrix we created above, **X**

```
In [6]: k_means.fit(X)
```

```
Out[6]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
   n_clusters=4, n_init=12, n_jobs=None, precompute_distances='auto',
   random_state=None, tol=0.0001, verbose=0)
```

Now let's grab the labels for each point in the model using KMeans' **.labels_** attribute and save it as **k_means_labels**

```
In [7]: k_means_labels = k_means.labels_
k_means_labels
```

```
Out[7]: array([0, 3, 3, ..., 1, 0, 0])
```

We will also get the coordinates of the cluster centers using KMeans' **.cluster_centers_** and save it as **k_means_cluster_centers**

```
In [8]: k_means_cluster_centers = k_means.cluster_centers_
k_means_cluster_centers
```



```
Out[8]: array([[-2.03743147, -0.99782524],
   [ 3.97334234,  3.98758687],
   [ 0.96900523,  0.98370298],
   [ 1.99741008, -3.01666822]])
```

Creating the Visual Plot

So now that we have the random data generated and the KMeans model initialized, let's plot them and see what it looks like!

Please read through the code and comments to understand how to plot the model.

```
In [9]: # Initialize the plot with the specified dimensions.
fig = plt.figure(figsize=(6, 4))

# Colors uses a color map, which will produce an array of colors based on
# the number of labels there are. We use set(k_means_labels) to get the
# unique labels.
colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k_means_labels)))))

# Create a plot
ax = fig.add_subplot(1, 1, 1)

# For loop that plots the data points and centroids.
# k will range from 0-3, which will match the possible clusters that each
# data point is in.
for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1, 1]])), colors):

    # Create a list of all data points, where the data poitns that are
    # in the cluster (ex. cluster 0) are labeled as true, else they are
    # labeled as false.
    my_members = (k_means_labels == k)

    # Define the centroid, or cluster center.
    cluster_center = k_means_cluster_centers[k]

    # Plots the datapoints with color col.
    ax.plot(X[my_members, 0], X[my_members, 1], 'w', markerfacecolor=col, marker='.')

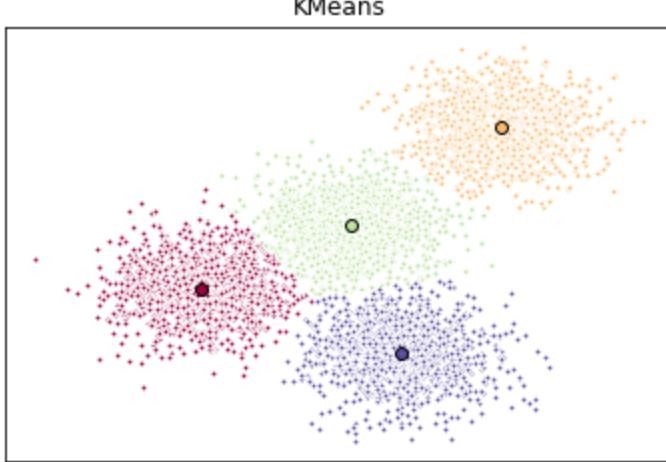
    # Plots the centroids with specified color, but with a darker outline
    ax.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col, markeredgewidth=2)

# Title of the plot
ax.set_title('KMeans')

# Remove x-axis ticks
ax.set_xticks(())

# Remove y-axis ticks
ax.set_yticks(())

# Show the plot
plt.show()
```



Practice

Try to cluster the above dataset into 3 clusters.

Notice: do not generate data again, use the same dataset as above.

In []: `# write your code here`

Double-click **here** for the solution.

Customer Segmentation with K-Means

Imagine that you have a customer dataset, and you need to apply customer segmentation on this historical data. Customer segmentation is the practice of partitioning a customer base into groups of individuals that have similar characteristics. It is a significant strategy as a business can target these specific groups of customers and effectively allocate marketing resources. For example, one group might contain customers who are high-profit and low-risk, that is, more likely to purchase products, or subscribe for a service. A business task is to retaining those customers. Another group might include customers from non-profit organizations. And so on.

Представьте, что у вас есть набор данных о клиентах, и вам нужно применить сегментацию клиентов к этим историческим данным. Сегментация клиентов — это практика разделения клиентской базы на группы лиц со схожими характеристиками. Это важная стратегия, поскольку бизнес может ориентироваться на эти конкретные группы клиентов и эффективно распределять маркетинговые ресурсы. Например, одна группа может содержать клиентов с высокой прибылью и низким уровнем риска, то есть с большей вероятностью купят продукты или подпишутся на услугу. Бизнес-задача состоит в том, чтобы удержать этих клиентов. Другая группа может включать клиентов из некоммерческих организаций. И так далее.

Load Data From CSV File

Before you can work with the data, you must use the URL to get the `Cust_Segmentation.csv`.

In [10]: `import pandas as pd
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()`

Out[10]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

Pre-processing

As you can see, **Address** in this dataset is a categorical variable. k-means algorithm isn't directly applicable to categorical variables because Euclidean distance function isn't really meaningful for discrete variables. So, lets drop this feature and run clustering.

In [11]:

```
df = cust_df.drop('Address', axis=1)
df.head()
```

Out[11]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	6.3
1	2	47	1	26	100	4.582	8.218	0.0	12.8
2	3	33	2	10	57	6.111	5.802	1.0	20.9
3	4	29	2	4	19	0.681	0.516	0.0	6.3
4	5	47	1	31	253	9.308	8.908	0.0	7.2

Normalizing over the standard deviation

Now let's normalize the dataset. But why do we need normalization in the first place? Normalization is a statistical method that helps mathematical-based algorithms to interpret features with different magnitudes and distributions equally. We use **StandardScaler()** to normalize our dataset.

In [12]:

```
from sklearn.preprocessing import StandardScaler
X = df.values[:,1:]
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

Out[12]:

```
array([[ 0.74291541,  0.31212243, -0.37878978, ..., -0.59048916,
       -0.52379654, -0.57652509],
       [ 1.48949049, -0.76634938,  2.5737211 , ...,  1.51296181,
       -0.52379654,  0.39138677],
       [-0.25251804,  0.31212243,  0.2117124 , ...,  0.80170393,
        1.90913822,  1.59755385],
       ...,
      [-1.24795149,  2.46906604, -1.26454304, ...,  0.03863257,
       1.90913822,  3.45892281],
      [-0.37694723, -0.76634938,  0.50696349, ..., -0.70147601,
       -0.52379654, -1.08281745],
      [ 2.1116364 , -0.76634938,  1.09746566, ...,  0.16463355,
       -0.52379654, -0.2340332 ]])
```

Modeling

In our example (if we didn't have access to the k-means algorithm), it would be the same as guessing that each customer group would have certain age, income, education, etc, with multiple tests and experiments. However, using the K-means clustering we can do all this process much easier.

Lets apply k-means on our dataset, and take look at cluster labels.

```
In [13]: clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

Insights

We assign the labels to each row in dataframe.

```
In [14]: df["Clus_km"] = labels  
df.head(5)
```

Out[14]:	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km
0	1	41	2	6	19	0.124	1.073	0.0	6.3	1
1	2	47	1	26	100	4.582	8.218	0.0	12.8	2
2	3	33	2	10	57	6.111	5.802	1.0	20.9	1
3	4	29	2	4	19	0.681	0.516	0.0	6.3	1
4	5	47	1	31	253	9.308	8.908	0.0	7.2	0

We can easily check the centroid values by averaging the features in each cluster.

```
In [15]: df.groupby('Clus_km').mean()
```

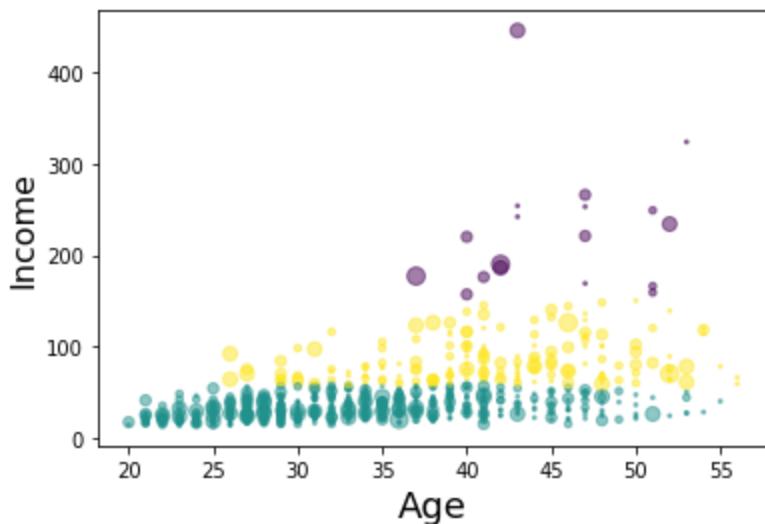
```
Out[15]:
```

Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Debtli
Clus_km								
0	410.166667	45.388889	2.666667	19.555556	227.166667	5.678444	10.907167	0.285714
1	432.006154	32.967692	1.613846	6.389231	31.204615	1.032711	2.108345	0.284658
2	403.780220	41.368132	1.961538	15.252747	84.076923	3.114412	5.770352	0.172414

Now, lets look at the distribution of customers based on their age and income:

```
In [16]:
```

```
area = np.pi * ( X[:, 1])**2  
plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)  
plt.xlabel('Age', fontsize=18)  
plt.ylabel('Income', fontsize=16)  
  
plt.show()
```

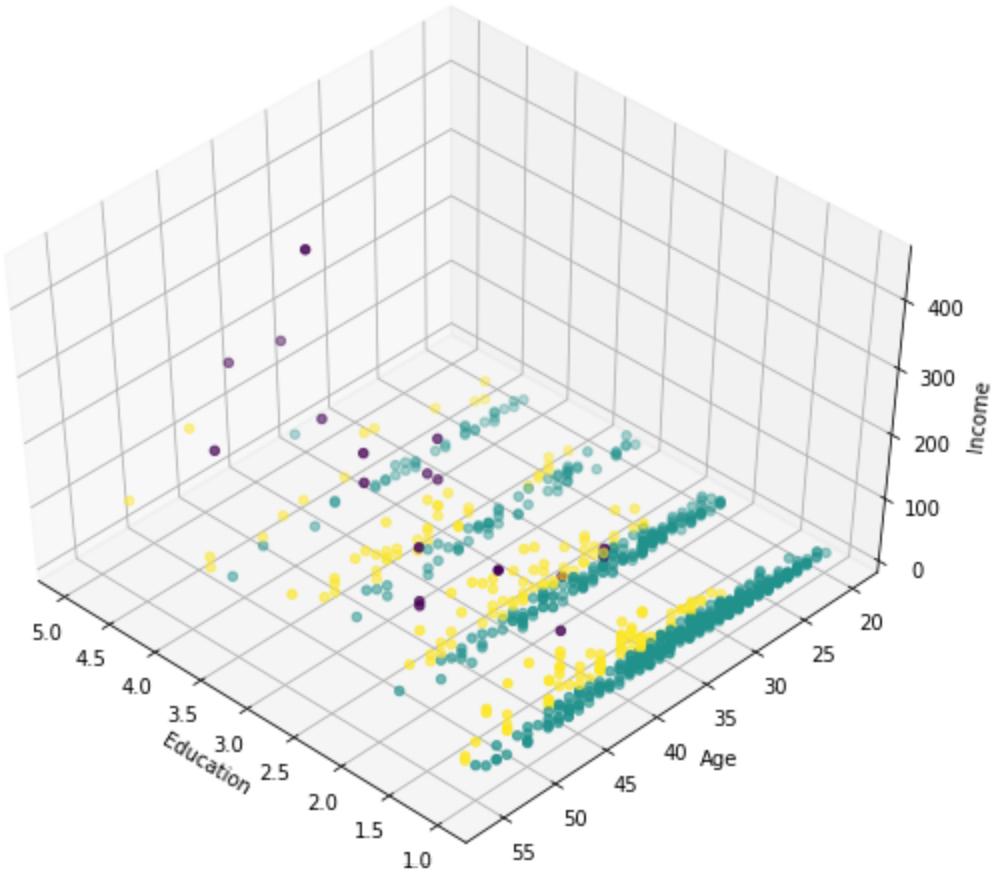


```
In [17]:
```

```
from mpl_toolkits.mplot3d import Axes3D  
fig = plt.figure(1, figsize=(8, 6))  
plt.clf()  
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)  
  
plt.cla()  
# plt.ylabel('Age', fontsize=18)  
# plt.xlabel('Income', fontsize=16)  
# plt.zlabel('Education', fontsize=16)  
ax.set_xlabel('Education')  
ax.set_ylabel('Age')  
ax.set_zlabel('Income')  
  
ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))
```

```
Out[17]:
```

```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x2c064970288>
```



k-means will partition your customers into mutually exclusive groups, for example, into 3 clusters. The customers in each cluster are similar to each other demographically. Now we can create a profile for each group, considering the common characteristics of each cluster. For example, the 3 clusters can be:

- AFFLUENT, EDUCATED AND OLD AGED
- MIDDLE AGED AND MIDDLE INCOME
- YOUNG AND LOW INCOME

Часть 4

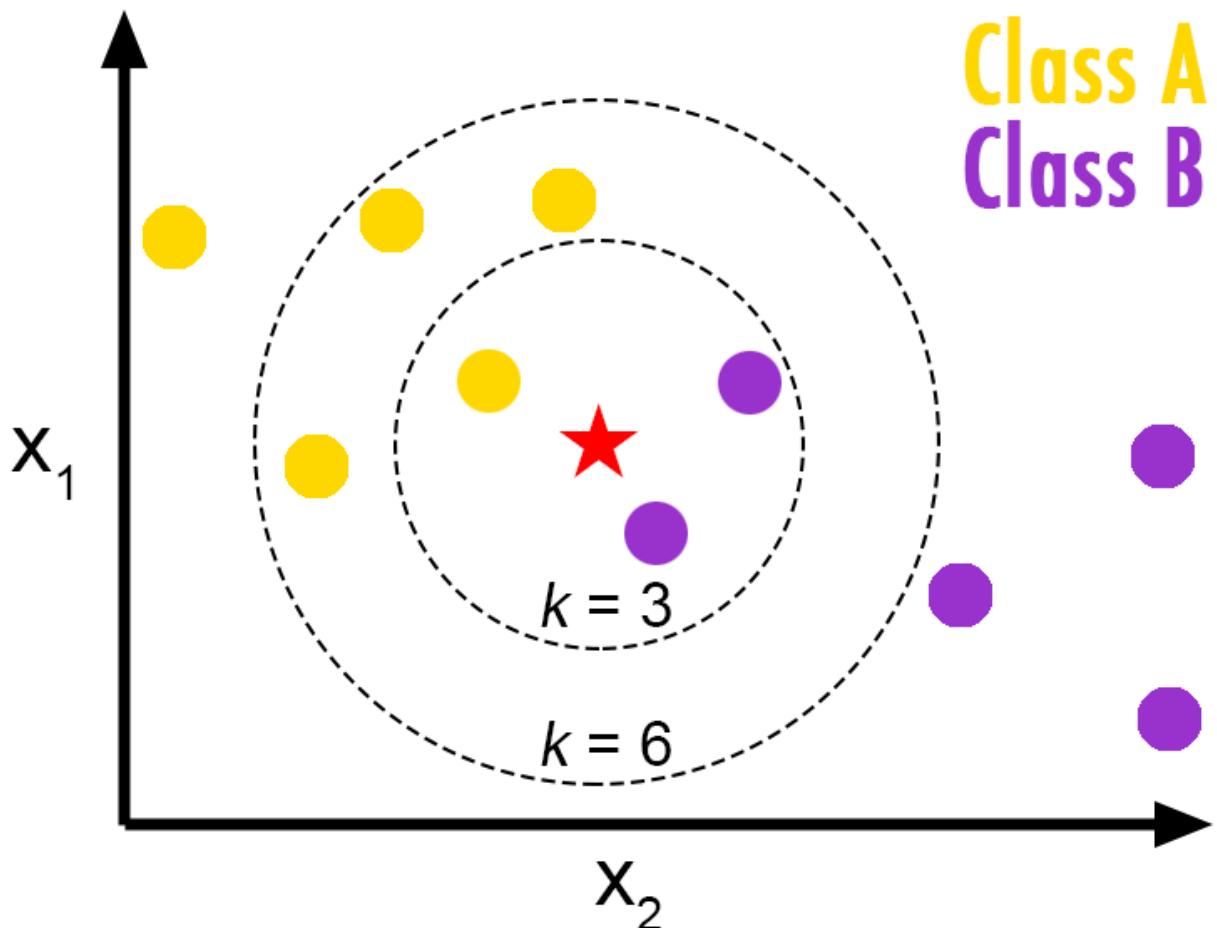
K-Nearest Neighbors

Теоретические сведения

In this Lab you will load a customer dataset, fit the data, and use K-Nearest Neighbors to predict a data point. But what is **K-Nearest Neighbors**?

K-Nearest Neighbors is an algorithm for supervised learning. Where the data is 'trained' with data points corresponding to their classification. Once a point is to be predicted, it takes into account the 'K' nearest points to it to determine it's classification.

Here's an visualization of the K-Nearest Neighbors algorithm.



In this case, we have data points of Class A and B. We want to predict what the star (test data point) is. If we consider a k value of 3 (3 nearest data points) we will obtain a prediction of Class B. Yet if we consider a k value of 6, we will obtain a prediction of Class A.

In this sense, it is important to consider the value of k. But hopefully from this diagram, you should get a sense of what the K-Nearest Neighbors algorithm is. It considers the 'K' Nearest Neighbors (points) when it predicts the classification of the test point.

Программа работы

1. About the dataset
2. Data Visualization and Analysis
3. Classification

Lets load required libraries

```
In [1]: import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

About the dataset

Imagine a telecommunications provider has segmented its customer base by service usage patterns, categorizing the customers into four groups. If demographic data can be used to predict group membership, the company can customize offers for individual prospective customers. It is a classification problem. That is, given the dataset, with predefined labels, we need to build a model to be used to predict class of a new or unknown case.

The example focuses on using demographic data, such as region, age, and marital, to predict usage patterns.

The target field, called **custcat**, has four possible values that correspond to the four customer groups, as follows: 1- Basic Service 2- E-Service 3- Plus Service 4- Total Service

Our objective is to build a classifier, to predict the class of unknown cases. We will use a specific type of classification called K nearest neighbour.

Представьте себе, что поставщик телекоммуникационных услуг сегментировал свою клиентскую базу по шаблонам использования услуг, разделив клиентов на четыре группы. Если демографические данные можно использовать для прогнозирования членства в группе, компания может настроить предложения для отдельных потенциальных клиентов. Это проблема классификации. То есть, учитывая набор данных с предопределенными метками, нам нужно построить модель, которая будет использоваться для прогнозирования класса нового или

неизвестного случая. В примере основное внимание уделяется использованию демографических данных, таких как регион, возраст и брак, для прогнозирования моделей использования. Целевое поле, называемое custcat, имеет четыре возможных значения, которые соответствуют четырем группам клиентов, а именно: 1 — базовая услуга 2 — электронная услуга 3 — дополнительная услуга 4 — общая услуга Наша цель — построить классификатор, чтобы предсказать класс неизвестных случаев. Мы будем использовать особый тип классификации, называемый К ближайшим соседом.

Load Data From CSV File

```
In [2]: df = pd.read_csv('teleCust1000t.csv')
df.head()
```

```
Out[2]:   region  tenure  age  marital  address  income  ed  employ  retire  gender  reside  custcat
0        2       13    44        1       9     64.0    4      5    0.0      0        2       1
1        3       11    33        1       7    136.0    5      5    0.0      0        6       4
2        3       68    52        1      24    116.0    1     29    0.0      1        2       3
3        2       33    33        0      12    33.0     2      0    0.0      1        1       1
4        2       23    30        1      9    30.0     1      2    0.0      0        4       3
```

Data Visualization and Analysis

Let's see how many of each class is in our data set

```
In [3]: df['custcat'].value_counts()
```

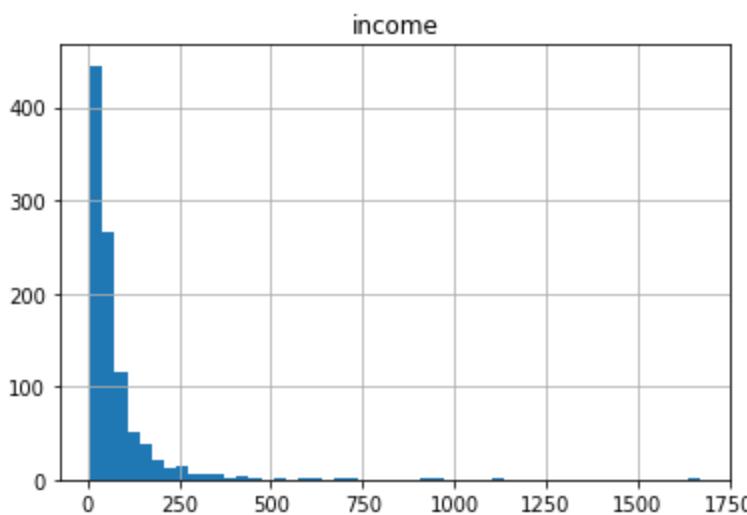
```
Out[3]: 3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

281 Plus Service, 266 Basic-service, 236 Total Service, and 217 E-Service customers

You can easily explore your data using visualization techniques:

```
In [4]: df.hist(column='income', bins=50)
```

```
Out[4]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000020D60763708>],  
              dtype=object)
```



Feature set

Lets define feature sets, X:

In [5]: `df.columns`

Out[5]: `Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed', 'employ', 'retire', 'gender', 'reside', 'custcat'], dtype='object')`

To use scikit-learn library, we have to convert the Pandas data frame to a Numpy array:

In [6]: `x = df[['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed', 'employ', 'retire']]
x[0:5]`

Out[6]: `array([[2., 13., 44., 1., 9., 64., 4., 5., 0., 0., 2.],
 [3., 11., 33., 1., 7., 136., 5., 5., 0., 0., 6.],
 [3., 68., 52., 1., 24., 116., 1., 29., 0., 1., 2.],
 [2., 33., 33., 0., 12., 33., 2., 0., 0., 1., 1.],
 [2., 23., 30., 1., 9., 30., 1., 2., 0., 0., 4.]])`

What are our labels?

In [7]: `y = df['custcat'].values
y[0:5]`

Out[7]: `array([1, 4, 3, 1, 3], dtype=int64)`

Normalize Data

Data Standardization give data zero mean and unit variance, it is good practice, especially for algorithms such as KNN which is based on distance of cases:

In [8]: `X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]`

Out[8]: `array([-0.02696767, -1.055125 , 0.18450456, 1.0100505 , -0.25303431,
 -0.12650641, 1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
 -0.23065004],
 [1.19883553, -1.14880563, -0.69181243, 1.0100505 , -0.4514148 ,
 0.54644972, 1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
 2.55666158],
 [1.19883553, 1.52109247, 0.82182601, 1.0100505 , 1.23481934,`

```
0.35951747, -1.36767088, 1.78752803, -0.22207644, 0.96655883,  
-0.23065004],  
[-0.02696767, -0.11831864, -0.69181243, -0.9900495 , 0.04453642,  
-0.41625141, -0.54919639, -1.09029981, -0.22207644, 0.96655883,  
-0.92747794],  
[-0.02696767, -0.58672182, -0.93080797, 1.0100505 , -0.25303431,  
-0.44429125, -1.36767088, -0.89182893, -0.22207644, -1.03459817,  
1.16300577]])
```

Train Test Split

Out of Sample Accuracy is the percentage of correct predictions that the model makes on data that that the model has NOT been trained on. Doing a train and test on the same dataset will most likely have low out-of-sample accuracy, due to the likelihood of being over-fit.

It is important that our models have a high, out-of-sample accuracy, because the purpose of any model, of course, is to make correct predictions on unknown data. So how can we improve out-of-sample accuracy? One way is to use an evaluation approach called Train/Test Split. Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set.

This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that have been used to train the data. It is more realistic for real world problems.

Точность вне выборки — это процент правильных прогнозов, которые модель обеспечивает на данных, которые НЕ использовались для обучения модели. Выполнение обучения и тестирования на одном и том же наборе данных, скорее всего, будет иметь низкую точность вне выборки из-за вероятности избыточного соответствия. Важно, чтобы модель имела высокую вневыборочную точность, потому что цель любой модели, конечно, состоит в том, чтобы делать правильные прогнозы на неизвестных данных. Один из способов улучшить точность вне выборки — использовать подход к оценке под названием Train/Test Split. Train/Test Split включает в себя разделение набора данных на наборы для обучения и тестирования соответственно, которые являются взаимоисключающими. После этого вы тренируетесь с тренировочным набором и тестируете с помощью тестового набора. Это обеспечит более точную оценку точности вне выборки, поскольку набор данных тестирования не является частью набора данных, который использовался для обучения данных.

```
In [9]: from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split( x, y, test_size=0.2, random_state=4  
print ('Train set:', X_train.shape, y_train.shape)  
print ('Test set:', X_test.shape, y_test.shape)
```

Train set: (800, 11) (800,)
Test set: (200, 11) (200,)

Classification

K nearest neighbor (KNN)

Import library

Classifier implementing the k-nearest neighbors vote.

```
In [10]: from sklearn.neighbors import KNeighborsClassifier
```

Training

Lets start the algorithm with k=4 for now:

```
In [11]:
```

```
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
Out[11]:
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                      metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                      weights='uniform')
```

Predicting

we can use the model to predict the test set:

```
In [12]:
```

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

```
Out[12]:
```

```
array([1, 1, 3, 2, 4], dtype=int64)
```

Accuracy evaluation

In multilabel classification, **accuracy classification score** is a function that computes subset accuracy. This function is equal to the jaccard_similarity_score function. Essentially, it calculates how closely the actual labels and predicted labels are matched in the test set.

```
In [13]:
```

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))

Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

Practice

Can you build the model again, but this time with k=6?

```
In [14]:
```

```
# write your code here
k = 6
neigh2 = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
yhat2 = neigh2.predict(X_test)

print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh2.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat2))

Train set Accuracy:  0.51625
Test set Accuracy:  0.32
```

Double-click **here** for the solution.

What about other K?

K in KNN, is the number of nearest neighbors to examine. It is supposed to be specified by the User. So, how can we choose right value for K? The general solution is to reserve a part of your data for testing the accuracy of the model. Then chose k =1, use the training part for modeling, and calculate the accuracy of prediction using all samples in your test set. Repeat this process, increasing the k, and see which k is the best for your model.

We can calculate the accuracy of KNN for different Ks.

In [15]:

```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = []
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)

    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

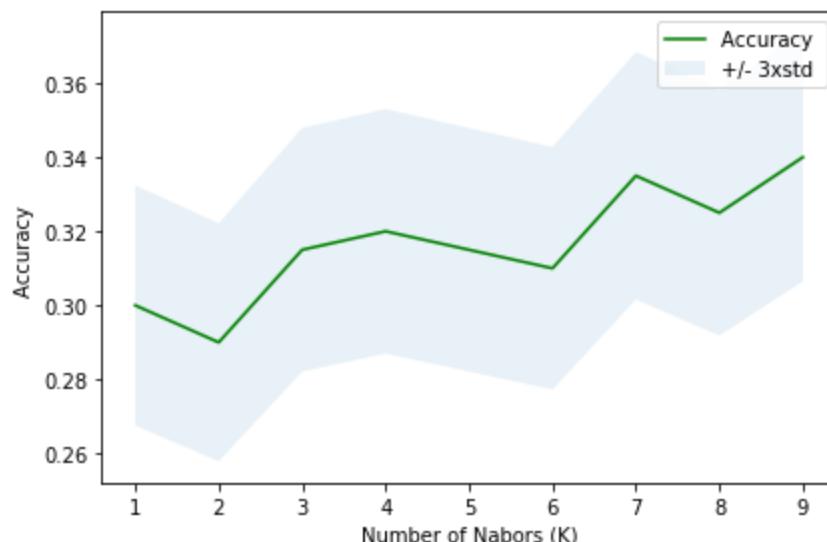
Out[15]:

```
array([0.3 , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

Plot model accuracy for Different number of Neighbors

In [16]:

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=0.10)
plt.legend(['Accuracy', '+/- 3xstd'])
plt.ylabel('Accuracy')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```



In [17]:

```
print("The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+1)
```

The best accuracy was with 0.34 with k= 9

Часть 5

Density-Based Clustering

Теоретические сведения

Most of the traditional clustering techniques, such as k-means, hierarchical and fuzzy clustering, can be used to group data without supervision.

However, when applied to tasks with arbitrary shape clusters, or clusters within cluster, the traditional techniques might be unable to achieve good results. That is, elements in the same cluster might not share enough similarity or the performance may be poor. Additionally, Density-based Clustering locates regions of high density that are separated from one another by regions of low density. Density, in this context, is defined as the number of points within a specified radius.

In this section, the main focus will be manipulating the data and properties of DBSCAN and observing the resulting clustering.

Большинство традиционных методов кластеризации, таких как метод k-средних, иерархическая и нечеткая кластеризация, можно использовать для группировки данных без контроля.

Однако, при применении к задачам с кластерами произвольной формы или кластерами внутри кластера традиционные методы могут не дать хороших результатов. То есть элементы в одном кластере могут иметь недостаточное сходство или производительность может быть низкой.

Кроме того, кластеризация на основе плотности находит области с высокой плотностью, которые отделены друг от друга областями с низкой плотностью. Плотность в данном контексте определяется как количество точек в пределах заданного радиуса.

В этом разделе основное внимание будет уделено управлению данными и свойствами DBSCAN и наблюдению за результирующей кластеризацией.

Программа работы

1. Clustering with Randomly Generated Data
 - A. Data generation
 - B. Modeling
 - C. Distinguishing Outliers
 - D. Data Visualization
2. Weather Station Clustering with DBSCAN & scikit-learn
 - A. Loading data
 - B. Overview data
 - C. Data cleaning

- D. Data selection
- E. Clustering
- F. Visualization of clusters based on location
- G. Clustering of stations based on their location, mean, max, and min Temperature
- H. Visualization of clusters based on location and Temperature

Import the following libraries:

- **numpy as np**
- **DBSCAN** from **sklearn.cluster**
- **make_blobs** from **sklearn.datasets.samples_generator**
- **StandardScaler** from **sklearn.preprocessing**
- **matplotlib.pyplot as plt**

Remember **%matplotlib inline** to display plots

In [1]:

```
# Notice: For visualization of map, you need basemap package.
# if you dont have basemap install on your machine, you can use the following line to in
#!pip install basemap --user
# from Anaconda Prompt
# conda install -c conda-forge basemap-data-hires
# Notice: you maight have to refresh your page and re-run the notebook after installatio
```

Requirement already satisfied: basemap in c:\users\1234\appdata\roaming\python\python37\site-packages (1.3.2)
Requirement already satisfied: numpy<1.23,>=1.21; python_version >= "3.7" in j:\anaconda3\lib\site-packages (from basemap) (1.21.5)
Requirement already satisfied: matplotlib<3.6,>=1.5; python_version >= "3.5" in j:\anacnda3\lib\site-packages (from basemap) (3.1.1)
Requirement already satisfied: six<1.16,>=1.10 in j:\anaconda3\lib\site-packages (from basemap) (1.12.0)
Requirement already satisfied: pyshp<2.2,>=1.2; python_version >= "2.7" in j:\anaconda3\lib\site-packages (from basemap) (2.1.3)
Requirement already satisfied: basemap-data<1.4,>=1.3.2 in c:\users\1234\appdata\roaming\python\python37\site-packages (from basemap) (1.3.2)
Requirement already satisfied: pyproj<3.4.0,>=1.9.3; python_version >= "3.5" in j:\anacnda3\lib\site-packages (from basemap) (3.2.1)
Requirement already satisfied: cycler>=0.10 in j:\anaconda3\lib\site-packages (from matplotlib<3.6,>=1.5; python_version >= "3.5"->basemap) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in j:\anaconda3\lib\site-packages (from matplotlib<3.6,>=1.5; python_version >= "3.5"->basemap) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!>=2.1.2,!>=2.1.6,>=2.0.1 in j:\anaconda3\lib\site-packages (from matplotlib<3.6,>=1.5; python_version >= "3.5"->basemap) (2.4.2)
Requirement already satisfied: python-dateutil>=2.1 in j:\anaconda3\lib\site-packages (from matplotlib<3.6,>=1.5; python_version >= "3.5"->basemap) (2.8.0)
Requirement already satisfied: certifi in j:\anaconda3\lib\site-packages (from pyproj<3.4.0,>=1.9.3; python_version >= "3.5"->basemap) (2019.9.11)
Requirement already satisfied: setuptools in j:\anaconda3\lib\site-packages (from kiwiso lver>=1.0.1->matplotlib<3.6,>=1.5; python_version >= "3.5"->basemap) (41.4.0)

In [2]:

```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs # https://stackoverflow.com/questions/65898399/n
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
%matplotlib inline
```

Data generation

The function below will generate the data points and requires these inputs:

- **centroidLocation**: Coordinates of the centroids that will generate the random data.
 - Example: input: [[4,3], [2,-1], [-1,4]]
- **numSamples**: The number of data points we want generated, split over the number of centroids (# of centroids defined in centroidLocation)
 - Example: 1500
- **clusterDeviation**: The standard deviation between the clusters. The larger the number, the further the spacing.
 - Example: 0.5

```
In [3]: def createDataPoints(centroidLocation, numSamples, clusterDeviation):  
    # Create random data and store in feature matrix X and response vector y.  
    X, y = make_blobs(n_samples=numSamples, centers=centroidLocation,  
                      cluster_std=clusterDeviation)  
  
    # Standardize features by removing the mean and scaling to unit variance  
    X = StandardScaler().fit_transform(X)  
    return X, y
```

Use **createDataPoints** with the **3 inputs** and store the output into variables **X** and **y**.

```
In [4]: X, y = createDataPoints([[4,3], [2,-1], [-1,4]], 1500, 0.5)
```

Modeling

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. This technique is one of the most common clustering algorithms which works based on density of object. The whole idea is that if a particular point belongs to a cluster, it should be near to lots of other points in that cluster.

It works based on two parameters: Epsilon and Minimum Points

Epsilon determine a specified radius that if includes enough number of points within, we call it dense area

minimumSamples determine the minimum number of data points we want in a neighborhood to define a cluster.

```
In [5]: epsilon = 0.3  
minimumSamples = 7  
db = DBSCAN(eps=epsilon, min_samples=minimumSamples).fit(X)  
labels = db.labels_
```

```
Out[5]: array([0, 0, 1, ..., 1, 1, 1], dtype=int64)
```

Distinguishing Outliers

Lets Replace all elements with 'True' in core_samples_mask that are in the cluster, 'False' if the points are outliers.

```
In [6]: # First, create an array of booleans using the labels from db.  
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)  
core_samples_mask[db.core_sample_indices_] = True  
core_samples_mask
```

```
Out[6]: array([ True,  True,  True, ...,  True,  True,  True])
```

```
In [7]: # Number of clusters in labels, ignoring noise if present.  
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)  
n_clusters_
```

```
Out[7]: 3
```

```
In [8]: # Remove repetition in labels by turning it into a set.  
unique_labels = set(labels)  
unique_labels
```

```
Out[8]: {0, 1, 2}
```

Data visualization

```
In [9]: # Create colors for the clusters.  
colors = plt.cm.Spectral(np.linspace(0, 1, len(unique_labels)))  
colors
```

```
Out[9]: array([[0.61960784, 0.00392157, 0.25882353, 1.          ],  
               [0.99807766, 0.99923106, 0.74602076, 1.          ],  
               [0.36862745, 0.30980392, 0.63529412, 1.          ]])
```

```
In [10]: # Plot the points with colors  
for k, col in zip(unique_labels, colors):  
    if k == -1:  
        # Black used for noise.  
        col = 'k'  
  
    class_member_mask = (labels == k)  
  
    # Plot the datapoints that are clustered  
    xy = X[class_member_mask & core_samples_mask]  
    plt.scatter(xy[:, 0], xy[:, 1], s=50, c=col, marker=u'o', alpha=0.5)  
  
    # Plot the outliers  
    xy = X[class_member_mask & ~core_samples_mask]  
    plt.scatter(xy[:, 0], xy[:, 1], s=50, c=col, marker=u'o', alpha=0.5)
```

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

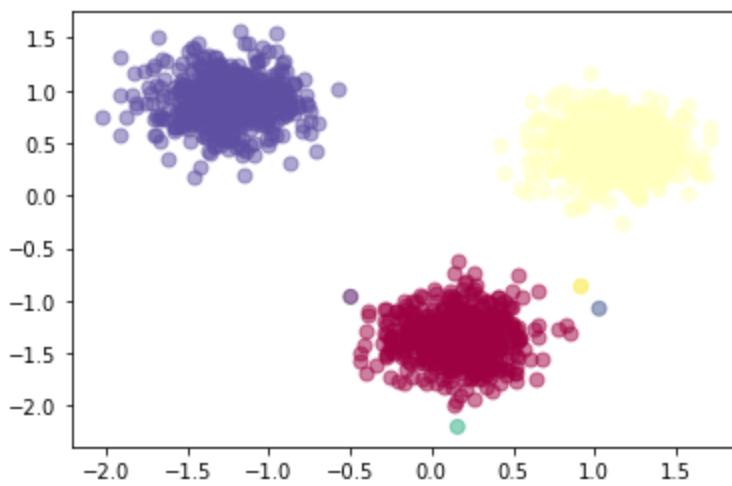
c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.

use the `*color*` keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.



Practice

To better understand differences between partitional and density-based clustering, try to cluster the above dataset into 3 clusters using k-Means.

Notice: do not generate data again, use the same dataset as above.

In [11]: `# write your code here`

Double-click **here** for the solution.

Weather Station Clustering using DBSCAN & scikit-learn

DBSCAN is specially very good for tasks like class identification on a spatial context. The wonderful attribute of DBSCAN algorithm is that it can find out any arbitrary shape cluster without getting affected by noise. For example, this following example cluster the location of weather stations in Canada. DBSCAN can be used here, for instance, to find the group of stations which show the same weather condition. As you can see, it not only finds different arbitrary shaped clusters, can find the denser part of data-centered samples by ignoring less-dense areas or noises.

let's start playing with the data. We will be working according to the following workflow:

About the dataset

Environment Canada Monthly Values for July - 2015

Name in the table	Meaning
<code>Stn_Name</code>	Station Name
<code>Lat</code>	Latitude (North+, degrees)
<code>Long</code>	Longitude (West -, degrees)
Prov	Province

Tm	Mean Temperature (°C)
DwTm	Days without Valid Mean Temperature
D	Mean Temperature difference from Normal (1981-2010) (°C)
Tx	Highest Monthly Maximum Temperature (°C)
DwTx	Days without Valid Maximum Temperature
Tn	Lowest Monthly Minimum Temperature (°C)
DwTn	Days without Valid Minimum Temperature
S	Snowfall (cm)
DwS	Days without Valid Snowfall
S%N	Percent of Normal (1981-2010) Snowfall
P	Total Precipitation (mm)
DwP	Days without Valid Precipitation
P%N	Percent of Normal (1981-2010) Precipitation
S_G	Snow on the ground at the end of the month (cm)
Pd	Number of days with Precipitation 1.0 mm or more
BS	Bright Sunshine (hours)
DwBS	Days without Valid Bright Sunshine
BS%	Percent of Normal (1981-2010) Bright Sunshine
HDD	Degree Days below 18 °C
CDD	Degree Days above 18 °C
Stn_No	Climate station identifier (first 3 digits indicate drainage basin, last 4 characters are for sorting alphabetically).
NA	Not Available

1-Download data

To download the data, we will use **!wget** to download it from IBM Object Storage.

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: [Sign up now for free](#)

```
In [12]: #!wget -O weather-stations20140101-20141231.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/DAT210/IBMCloud/weather-stations20140101-20141231.csv
```

2- Load the dataset

We will import the .csv then we creates the columns for year, month and day.

```
In [11]: import csv
import pandas as pd
import numpy as np

filename='weather-stations20140101-20141231.csv'

#Read csv
pdf = pd.read_csv(filename)
pdf.head(5)
```

Out[11]:

	Stn_Name	Long	Prov	Tm	DwTm	D	Tx	DwTx	Tn	...	DwP	P%	N	S_G	Pd	BS	DwBS	S%	HDD	CDD	Stn_No
0	CHEMAINUS	123.742	BC	8.2	0.0	NaN	13.5	0.0	1.0	...	0.0	NaN	0.0	12.0	NaN	NaN	NaN	273.3	0.0	1011500	
1	COWICHAN FORESTRY	124.133	BC	7.0	0.0	3.0	15.0	0.0	-3.0	...	0.0	104.0	0.0	12.0	NaN	NaN	NaN	307.0	0.0	1012040	
2	LAKE COWICHAN	124.052	BC	6.8	13.0	2.8	16.0	9.0	-2.5	...	9.0	NaN	NaN	11.0	NaN	NaN	NaN	168.1	0.0	1012055	
3	DISCOVERY ISLAND	123.226	BC	NaN	NaN	NaN	12.5	0.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1012475	
4	DUNCAN	123.728	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573	
KELVIN CREEK	123.735	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573		

5 rows × 25 columns

3-Cleaning

Lets remove rows that don't have any value in the **Tm** field.

```
In [12]: pdf = pdf[pd.notnull(pdf["Tm"])]
pdf = pdf.reset_index(drop=True)
pdf.head(5)
```

Out[12]:

	Stn_Name	Long	Prov	Tm	DwTm	D	Tx	DwTx	Tn	...	DwP	P%	N	S_G	Pd	BS	DwBS	S%	HDD	CDD	Stn_No
0	CHEMAINUS	123.742	BC	8.2	0.0	NaN	13.5	0.0	1.0	...	0.0	NaN	0.0	12.0	NaN	NaN	NaN	273.3	0.0	1011500	
1	COWICHAN FORESTRY	124.133	BC	7.0	0.0	3.0	15.0	0.0	-3.0	...	0.0	104.0	0.0	12.0	NaN	NaN	NaN	307.0	0.0	1012040	
2	LAKE COWICHAN	124.052	BC	6.8	13.0	2.8	16.0	9.0	-2.5	...	9.0	NaN	NaN	11.0	NaN	NaN	NaN	168.1	0.0	1012055	
3	DUNCAN	123.728	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573	
KELVIN CREEK	123.735	BC	7.7	2.0	3.4	14.5	2.0	-1.0	...	2.0	NaN	NaN	11.0	NaN	NaN	NaN	267.7	0.0	1012573		
4	ESQUIMALT HARBOUR	123.430	BC	8.8	0.0	NaN	13.1	0.0	1.9	...	8.0	NaN	NaN	12.0	NaN	NaN	NaN	258.6	0.0	1012710	

5 rows × 25 columns

4-Visualization

Visualization of stations on map using basemap package. The matplotlib basemap toolkit is a library for plotting 2D data on maps in Python. Basemap does not do any plotting on its own, but provides the facilities to transform coordinates to a map projections.

Please notice that the size of each data points represents the average of maximum temperature for each station in a year.

```
In [15]: # https://stackoverflow.com/questions/52295117/basemap-import-error-in-pycharm-keyerror-
# https://stackoverflow.com/questions/52911232/basemap-library-using-anaconda-jupyter-notebook
import os
os.environ["PROJ_LIB"] = "J:\Anaconda3\Library\share"; #fixr

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
```

```
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

llon=-140
ulon=-50
llat=40
ulat=65

pdf = pdf[(pdf['Long'] > llon) & (pdf['Long'] < ulon) & (pdf['Lat'] > llat) & (pdf['Lat'] < ulat)]

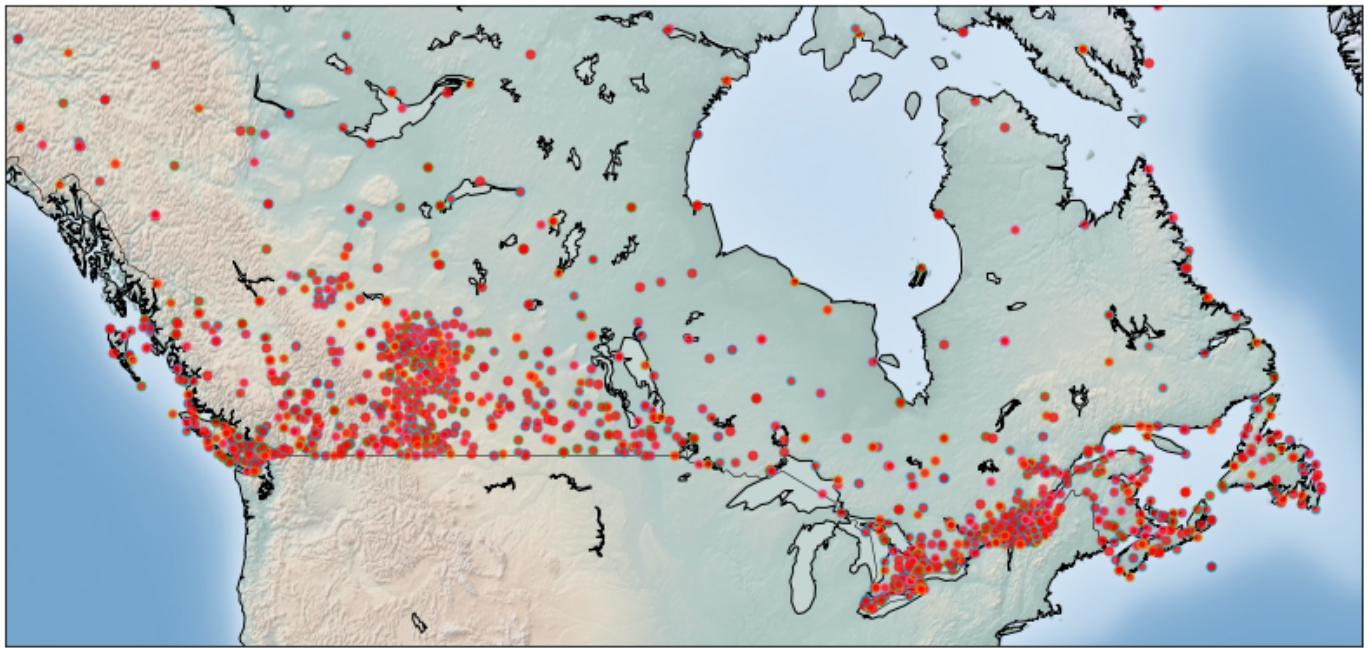
my_map = Basemap(projection='merc',
                  resolution = 'l', area_thresh = 1000.0,
                  llcrnrlon=llon, llcrnrlat=llat, #min longitude (llcrnrlon) and latitude (llcrnrlat)
                  urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (urcrnrlat)

my_map.drawcoastlines()
my_map.drawcountries()
# my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To collect data based on stations

xs,ys = my_map(np.asarray(pdf.Long), np.asarray(pdf.Lat))
pdf['xm']= xs.tolist()
pdf['ym'] =ys.tolist()

#Visualization1
for index,row in pdf.iterrows():
    x,y = my_map(row.Long, row.Lat)
    my_map.plot(xm, ym,markerfacecolor =([1,0,0]), marker='o', markersize= 5, alpha = 0.5)
    plt.text(x,y,stn)
plt.show()
```



5- Clustering of stations based on their location i.e. Lat & Lon

DBSCAN from sklearn library can runs DBSCAN clustering from vector array or distance matrix. In our case, we pass it the Numpy array Clus_dataSet to find core samples of high density and expands clusters from them.

In [16]: `from sklearn.cluster import DBSCAN`

```

import sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN
db = DBSCAN(eps=0.15, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"] = labels

realClusterNum = len(set(labels)) - (1 if -1 in labels else 0)
clusterNum = len(set(labels))

# A sample of clusters
pdf[["Stn_Name", "Tx", "Tm", "Clus_Db"]].head(5)

```

Out[16]:

	Stn_Name	Tx	Tm	Clus_Db
0	CHEMAINUS	13.5	8.2	0
1	COWICHAN LAKE FORESTRY	15.0	7.0	0
2	LAKE COWICHAN	16.0	6.8	0
3	DUNCAN KELVIN CREEK	14.5	7.7	0
4	ESQUIMALT HARBOUR	13.1	8.8	0

As you can see for outliers, the cluster label is -1

In [17]:

```
set(labels)
```

Out[17]:

```
{-1, 0, 1, 2, 3, 4}
```

6- Visualization of clusters based on location

Now, we can visualize the clusters using basemap:

In [18]:

```

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
                 resolution = 'l', area_thresh = 1000.0,
                 llcrnrlon=lllon, llcrnrlat=lllat, #min longitude (llcrnrlon) and latitude (llcrnrlat)
                 urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (urcrnrlat)

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map

```

```

colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))

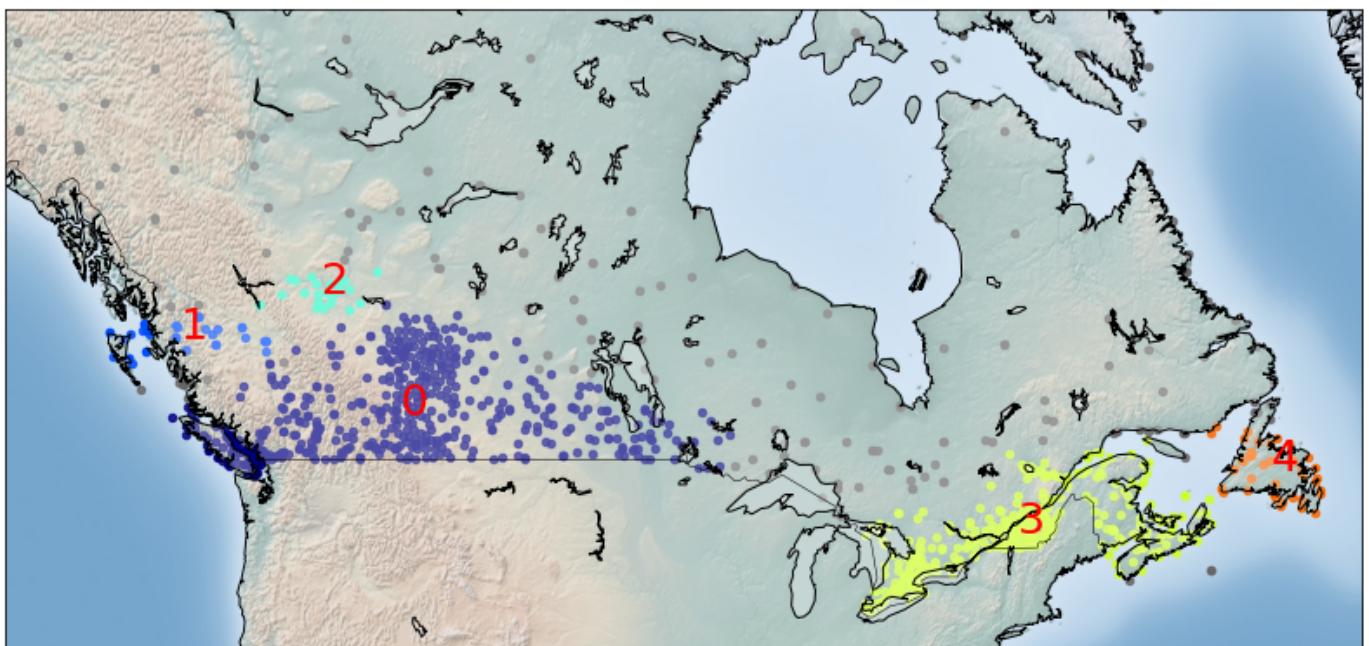
#Visualization1
for clust_number in set(labels):
    c=(([0.4,0.4,0.4]) if clust_number == -1 else colors[np.int(clust_number)])
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color =c, marker='o', s= 20, alpha = 0.8
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red')
        print ("Cluster "+str(clust_number)+', Avg Temp: '+ str(np.mean(clust_set.Tm)))

```

```

C:\Users\1234\AppData\Local\Temp\ipykernel_21284/3325651678.py:25: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
c=(([0.4,0.4,0.4]) if clust_number == -1 else colors[np.int(clust_number)])
Cluster 0, Avg Temp: -5.538747553816051
Cluster 1, Avg Temp: 1.9526315789473685
Cluster 2, Avg Temp: -9.195652173913045
Cluster 3, Avg Temp: -15.300833333333333
Cluster 4, Avg Temp: -7.769047619047619

```



7- Clustering of stations based on their location, mean, max, and min Temperature

In this section we re-run DBSCAN, but this time on a 5-dimensional dataset:

In [19]:

```

from sklearn.cluster import DBSCAN
import sklearn.utils
from sklearn.preprocessing import StandardScaler
sklearn.utils.check_random_state(1000)
Clus_dataSet = pdf[['xm','ym','Tx','Tm','Tn']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

# Compute DBSCAN

```

```

db = DBSCAN(eps=0.3, min_samples=10).fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
pdf["Clus_Db"] = labels

realClusterNum = len(set(labels)) - (1 if -1 in labels else 0)
clusterNum = len(set(labels))

# A sample of clusters
pdf[["Stn_Name", "Tx", "Tm", "Clus_Db"]].head(5)

```

Out[19]:

	Stn_Name	Tx	Tm	Clus_Db
0	CHEMAINUS	13.5	8.2	0
1	COWICHAN LAKE FORESTRY	15.0	7.0	0
2	LAKE COWICHAN	16.0	6.8	0
3	DUNCAN KELVIN CREEK	14.5	7.7	0
4	ESQUIMALT HARBOUR	13.1	8.8	0

8- Visualization of clusters based on location and Temperature

In [20]:

```

from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
from pylab import rcParams
%matplotlib inline
rcParams['figure.figsize'] = (14,10)

my_map = Basemap(projection='merc',
                  resolution = 'l', area_thresh = 1000.0,
                  llcrnrlon=lllon, llcrnrlat=lllat, #min longitude (llcrnrlon) and latitude (llcrnrlat)
                  urcrnrlon=ulon, urcrnrlat=ulat) #max longitude (urcrnrlon) and latitude (urcrnrlat)

my_map.drawcoastlines()
my_map.drawcountries()
#my_map.drawmapboundary()
my_map.fillcontinents(color = 'white', alpha = 0.3)
my_map.shadedrelief()

# To create a color map
colors = plt.get_cmap('jet')(np.linspace(0.0, 1.0, clusterNum))

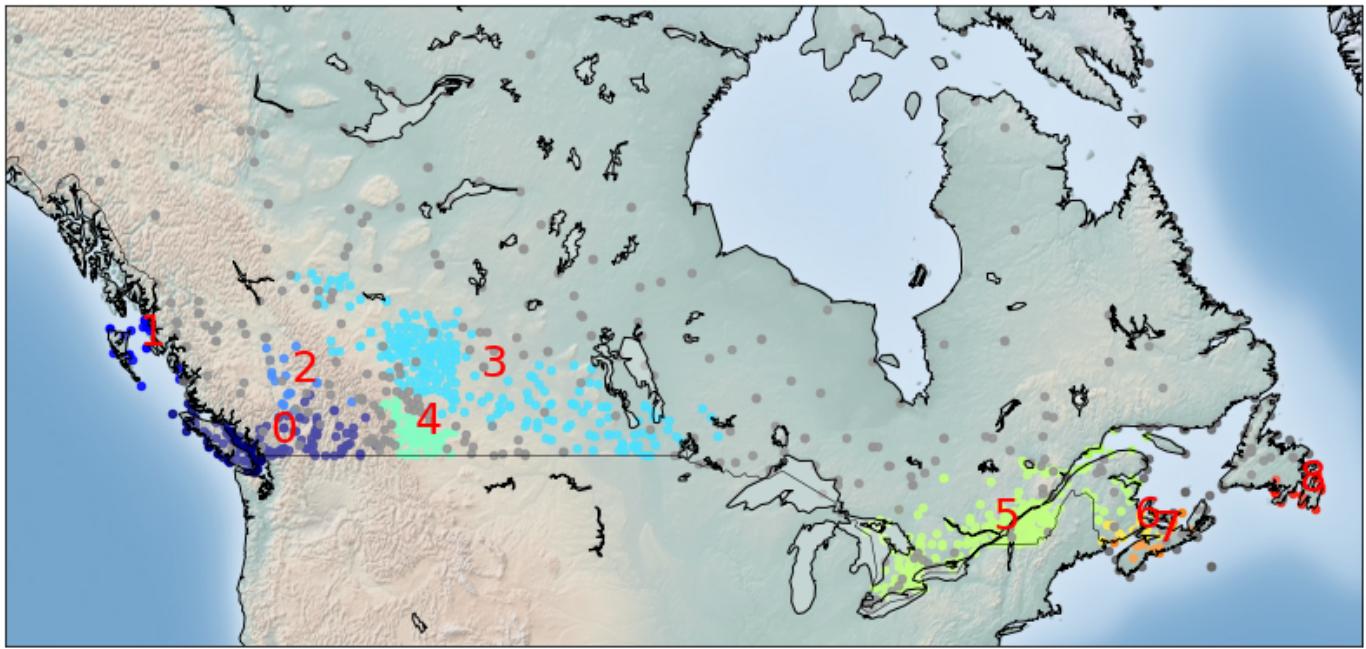
#Visualization
for clust_number in set(labels):
    c=((0.4,0.4,0.4)) if clust_number == -1 else colors[int(clust_number)])
    clust_set = pdf[pdf.Clus_Db == clust_number]
    my_map.scatter(clust_set.xm, clust_set.ym, color=c, marker='o', s= 20, alpha = 0.8
    if clust_number != -1:
        cenx=np.mean(clust_set.xm)
        ceny=np.mean(clust_set.ym)
        plt.text(cenx,ceny,str(clust_number), fontsize=25, color='red')
        print ("Cluster "+str(clust_number)+', Avg Temp: '+ str(np.mean(clust_set.Tm)))

```

C:\Users\1234\AppData\Local\Temp\ipykernel_21284/3325651678.py:25: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int`

by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>

```
c=(([0.4,0.4,0.4]) if clust_number == -1 else colors[np.int(clust_number)])  
Cluster 0, Avg Temp: 6.2211920529801334  
Cluster 1, Avg Temp: 6.7900000000000001  
Cluster 2, Avg Temp: -0.49411764705882355  
Cluster 3, Avg Temp: -13.877209302325586  
Cluster 4, Avg Temp: -4.186274509803922  
Cluster 5, Avg Temp: -16.301503759398482  
Cluster 6, Avg Temp: -13.599999999999998  
Cluster 7, Avg Temp: -9.753333333333334  
Cluster 8, Avg Temp: -4.258333333333334
```



In []:

Часть 6

Hierarchical Clustering

Теоретические сведения

Иерархическая кластеризация (также графовые алгоритмы кластеризации и иерархический кластерный анализ) — совокупность алгоритмов упорядочивания данных, направленных на создание иерархии (дерева) вложенных кластеров. Выделяют два класса методов иерархической кластеризации:

Агломеративные методы (англ. agglomerative): новые кластеры создаются путем объединения более мелких кластеров и, таким образом, дерево создается от листьев к стволу;

Дивизивные или дивизионные методы (англ. divisive): новые кластеры создаются путем деления более крупных кластеров на более мелкие и, таким образом, дерево создается от ствола к листьям.

Алгоритмы иерархической кластеризации предполагают, что анализируемое множество объектов характеризуется определённой степенью связности.

Программа работы

1. Hierarchical Clustering - Agglomerative
 - A. Generating Random Data
 - B. Agglomerative Clustering
 - C. Dendrogram Associated for the Agglomerative Hierarchical Clustering
2. Clustering on the Vehicle Dataset
 - A. Data Cleaning
 - B. Clustering Using Scipy
 - C. Clustering using scikit-learn

Hierarchical Clustering - Agglomerative

We will be looking at a clustering technique, which is **Agglomerative Hierarchical Clustering**. Remember that agglomerative is the bottom up approach.

In this lab, we will be looking at Agglomerative clustering, which is more popular than Divisive

clustering.

We will also be using Complete Linkage as the Linkage Criteria.

In [28]:

```
import numpy as np
import pandas as pd
from scipy import ndimage
from scipy.cluster import hierarchy
from scipy.spatial import distance_matrix
from matplotlib import pyplot as plt
from sklearn import manifold, datasets
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
```

Generating Random Data

We will be generating a set of data using the **make_blobs** class.

Input these parameters into make_blobs:

- **n_samples**: The total number of points equally divided among clusters.
 - Choose a number from 10-1500
- **centers**: The number of centers to generate, or the fixed center locations.
 - Choose arrays of x,y coordinates for generating the centers. Have 1-10 centers (ex. centers=[[1,1], [2,5]])
- **cluster_std**: The standard deviation of the clusters. The larger the number, the further apart the clusters
 - Choose a number between 0.5-1.5

Save the result to **X1** and **y1**.

In [2]:

```
X1, y1 = make_blobs(n_samples=50, centers=[[4,4], [-2, -1], [1, 1], [10, 4]], cluster_std
```

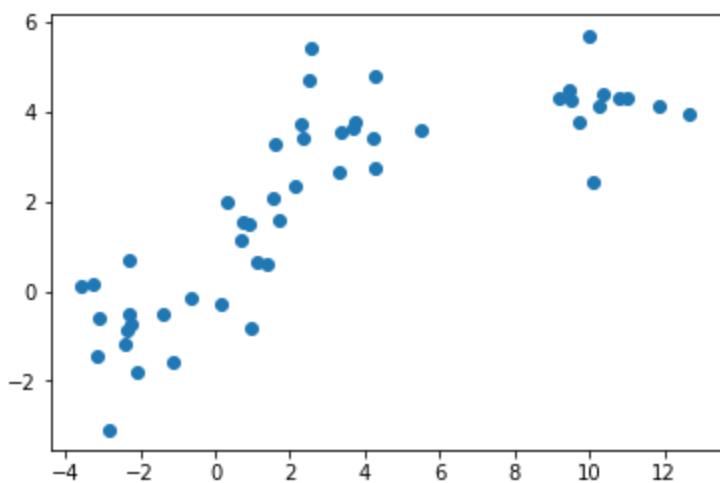
Plot the scatter plot of the randomly generated data

In [3]:

```
plt.scatter(X1[:, 0], X1[:, 1], marker='o')
```

Out[3]:

```
<matplotlib.collections.PathCollection at 0x24504d364c8>
```



Agglomerative Clustering

We will start by clustering the random data points we just created.

The **Agglomerative Clustering** class will require two inputs:

- **n_clusters**: The number of clusters to form as well as the number of centroids to generate.
 - Value will be: 4
- **linkage**: Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.
 - Value will be: 'complete'
 - **Note**: It is recommended you try everything with 'average' as well

Save the result to a variable called **agglom**

```
In [4]: agglom = AgglomerativeClustering(n_clusters = 4, linkage = 'average')
```

Fit the model with **X2** and **y2** from the generated data above.

```
In [29]: agglom.fit(X1, y1)
```

```
J:\Anaconda3\lib\site-packages\sklearn\cluster\hierarchical.py:464: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.  
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

```
    children_ = out[:, :2].astype(np.int, copy=False)
```

```
Out[29]: AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',  
                                 connectivity=None, distance_threshold=None,  
                                 linkage='complete', memory=None, n_clusters=6,  
                                 pooling_func='deprecated')
```

Run the following code to show the clustering!

Remember to read the code and comments to gain more understanding on how the plotting works.

```
In [6]: # Create a figure of size 6 inches by 4 inches.  
plt.figure(figsize=(6,4))
```

```

# These two lines of code are used to scale the data points down,
# Or else the data points will be scattered very far apart.

# Create a minimum and maximum range of X1.
x_min, x_max = np.min(X1, axis=0), np.max(X1, axis=0)

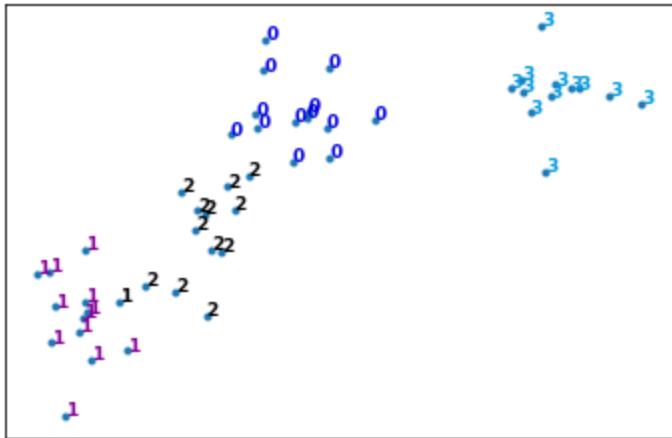
# Get the average distance for X1.
X1 = (X1 - x_min) / (x_max - x_min)

# This loop displays all of the datapoints.
for i in range(X1.shape[0]):
    # Replace the data points with their respective cluster value
    # (ex. 0) and is color coded with a colormap (plt.cm.spectral)
    plt.text(X1[i, 0], X1[i, 1], str(y1[i]),
             color=plt.cm.nipy_spectral(aglom.labels_[i] / 10.),
             fontdict={'weight': 'bold', 'size': 9})

# Remove the x ticks, y ticks, x and y axis
plt.xticks([])
plt.yticks([])
# plt.axis('off')

# Display the plot of the original data before clustering
plt.scatter(X1[:, 0], X1[:, 1], marker='.')
# Display the plot
plt.show()

```



Dendrogram Associated for the Agglomerative Hierarchical Clustering

Remember that a **distance matrix** contains the **distance from each point to every other point of a dataset**.

Use the function **distance_matrix**, which requires **two inputs**. Use the Feature Matrix, **X2** as both inputs and save the distance matrix to a variable called **dist_matrix**

Remember that the distance values are symmetric, with a diagonal of 0's. This is one way of making sure your matrix is correct.

(print out dist_matrix to make sure it's correct)

In [7]:

```
dist_matrix = distance_matrix(X1, X1)
print(dist_matrix)
```

```
[0.          0.14524927  0.25591624 ... 1.01198044  0.38518978  0.4969025 ]
```

```
[0.14524927 0.          0.39492083 ... 1.12044274 0.52603268 0.63686534]
[0.25591624 0.39492083 0.          ... 0.78347041 0.13158973 0.24194949]
...
[1.01198044 1.12044274 0.78347041 ... 0.          0.69475013 0.6107001 ]
[0.38518978 0.52603268 0.13158973 ... 0.69475013 0.          0.11198439]
[0.4969025  0.63686534 0.24194949 ... 0.6107001  0.11198439 0.          ]]
```

Using the **linkage** class from hierarchy, pass in the parameters:

- The distance matrix
- 'complete' for complete linkage

Save the result to a variable called **Z**

```
In [8]: Z = hierarchy.linkage(dist_matrix, 'complete')
```

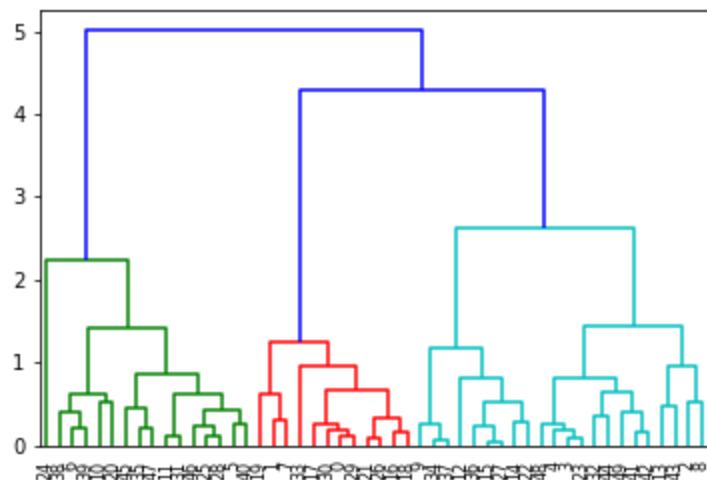
```
J:\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: ClusterWarning: scipy.cluster: T
he symmetric non-negative hollow observation matrix looks suspiciously like an uncondens
ed distance matrix
"""\Entry point for launching an IPython kernel.
```

A Hierarchical clustering is typically visualized as a dendrogram as shown in the following cell. Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged, where cities are viewed as singleton clusters. By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering.

Next, we will save the dendrogram to a variable called **dendro**. In doing this, the dendrogram will also be displayed. Using the **dendrogram** class from hierarchy, pass in the parameter:

- **Z**

```
In [9]: dendro = hierarchy.dendrogram(Z)
```



Practice

We used **complete** linkage for our case, change it to **average** linkage to see how the dendrogram changes.

```
In [10]: # write your code here
```

Clustering on Vehicle dataset

Imagine that an automobile manufacturer has developed prototypes for a new vehicle. Before introducing the new model into its range, the manufacturer wants to determine which existing vehicles on the market are most like the prototypes--that is, how vehicles can be grouped, which group is the most similar with the model, and therefore which models they will be competing against.

Our objective here, is to use clustering methods, to find the most distinctive clusters of vehicles. It will summarize the existing vehicles and help manufacturers to make decision about the supply of new models.

Download data

To download the data, we will use **!wget** to download it from IBM Object Storage.

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: [Sign up now for free](#)

```
In [11]: #!wget -O cars_clus.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-dat
```

Read data

lets read dataset to see what features the manufacturer has collected about the existing models.

```
In [12]: filename = 'cars_clus.csv'

#Read csv
pdf = pd.read_csv(filename)
print ("Shape of dataset: ", pdf.shape)

pdf.head(5)
```

Shape of dataset: (159, 16)

```
Out[12]:   manufact  model  sales  resale  type  price  engine_s  horsepow  wheelbas  width  length  curl
          0      Acura  Integra  16.919  16.360  0.000  21.500     1.800    140.000   101.200   67.300  172.400
          1      Acura       TL  39.384  19.875  0.000  28.400     3.200    225.000   108.100   70.300  192.900
          2      Acura       CL  14.114  18.225  0.000     null     3.200    225.000   106.900   70.600  192.000
          3      Acura       RL   8.588  29.725  0.000  42.000     3.500    210.000   114.600   71.400  196.600
          4      Audi        A4  20.397  22.255  0.000  23.990     1.800    150.000   102.600   68.200  178.000
```

The feature sets include price in thousands (price), engine size (engine_s), horsepower (horsepow), wheelbase (wheelbas), width (width), length (length), curb weight (curb_wgt), fuel capacity (fuel_cap) and fuel efficiency (mpg).

Data Cleaning

Lets simply clear the dataset by dropping the rows that have null value:

```
In [13]: print ("Shape of dataset before cleaning: ", pdf.size)
pdf[[ 'sales', 'resale', 'type', 'price', 'engine_s',
      'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
      'mpg', 'lnsales']] = pdf[['sales', 'resale', 'type', 'price', 'engine_s',
      'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'fuel_cap',
      'mpg', 'lnsales']].apply(pd.to_numeric, errors='coerce')
pdf = pdf.dropna()
pdf = pdf.reset_index(drop=True)
print ("Shape of dataset after cleaning: ", pdf.size)
pdf.head(5)
```

Shape of dataset before cleaning: 2544

Shape of dataset after cleaning: 1872

```
Out[13]:   manufact model sales resale type price engine_s horsepow wheelbas width length curb_wg
0     Acura Integra  16.919  16.360  0.0  21.50      1.8     140.0    101.2   67.3   172.4   2.63
1     Acura      TL  39.384  19.875  0.0  28.40      3.2     225.0    108.1   70.3   192.9   3.51
2     Acura      RL   8.588  29.725  0.0  42.00      3.5     210.0    114.6   71.4   196.6   3.85
3     Audi       A4  20.397  22.255  0.0  23.99      1.8     150.0    102.6   68.2   178.0   2.99
4     Audi       A6  18.780  23.555  0.0  33.95      2.8     200.0    108.7   76.1   192.0   3.56
```

Feature selection

Lets select our feature set:

```
In [14]: featureset = pdf[['engine_s', 'horsepow', 'wheelbas', 'width', 'length', 'curb_wgt', 'f
```

Normalization

Now we can normalize the feature set. **MinMaxScaler** transforms features by scaling each feature to a given range. It is by default (0, 1). That is, this estimator scales and translates each feature individually such that it is between zero and one.

```
In [15]: from sklearn.preprocessing import MinMaxScaler
x = featureset.values #returns a numpy array
min_max_scaler = MinMaxScaler()
feature_mtx = min_max_scaler.fit_transform(x)
feature_mtx [0:5]
```

```
Out[15]: array([[0.11428571, 0.21518987, 0.18655098, 0.28143713, 0.30625832,
      0.2310559 , 0.13364055, 0.43333333],
     [0.31428571, 0.43037975, 0.3362256 , 0.46107784, 0.5792277 ,
      0.50372671, 0.31797235, 0.33333333],
     [0.35714286, 0.39240506, 0.47722343, 0.52694611, 0.62849534,
      0.60714286, 0.35483871, 0.23333333],
     [0.11428571, 0.24050633, 0.21691974, 0.33532934, 0.38082557,
      0.34254658, 0.28110599, 0.4       ],
     [0.25714286, 0.36708861, 0.34924078, 0.80838323, 0.56724368,
      0.5173913 , 0.37788018, 0.23333333]])
```

Clustering using Scipy

In this part we use Scipy package to cluster the dataset:

First, we calculate the distance matrix.

```
In [16]: import scipy
leng = feature_mtx.shape[0]
D = scipy.zeros([leng,leng])
for i in range(leng):
    for j in range(leng):
        D[i,j] = scipy.spatial.distance.euclidean(feature_mtx[i], feature_mtx[j])
```

In agglomerative clustering, at each iteration, the algorithm must update the distance matrix to reflect the distance of the newly formed cluster with the remaining clusters in the forest. The following methods are supported in Scipy for calculating the distance between the newly formed cluster and each:

- single
- complete
- average
- weighted
- centroid

We use **complete** for our case, but feel free to change it to see how the results change.

```
In [17]: import pylab
import scipy.cluster.hierarchy
Z = hierarchy.linkage(D, 'complete')
```

J:\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: ClusterWarning: scipy.cluster: The symmetric non-negative hollow observation matrix looks suspiciously like an uncondensed distance matrix

This is separate from the ipykernel package so we can avoid doing imports until

Essentially, Hierarchical clustering does not require a pre-specified number of clusters. However, in some applications we want a partition of disjoint clusters just as in flat clustering. So you can use a cutting line:

```
In [18]: from scipy.cluster.hierarchy import fcluster
max_d = 3
clusters = fcluster(Z, max_d, criterion='distance')
clusters
```

```
Out[18]: array([ 1,  5,  5,  6,  5,  4,  6,  5,  5,  5,  5,  5,  5,  4,  4,
      5,  1,  6,  6, 10,  9,  8,
      9,  3,  5,  1,  7,  6,  5,  3,  5,  3,  8,  7,  9,  2,  6,  6,  5,
      4,  2,  1,  6,  5,  2,  7,  5,  5,  5,  4,  4,  3,  2,  6,  6,  5,
      7,  4,  7,  6,  6,  5,  3,  5,  5,  6,  5,  4,  4,  1,  6,  5,  5,
      5,  6,  4,  5,  4,  1,  6,  5,  6,  6,  5,  5,  5,  7,  7,  7,  2,
      2,  1,  2,  6,  5,  1,  1,  7,  8,  1,  1,  6,  1,  1],  
      dtype=int32)
```

Also, you can determine the number of clusters directly:

```
In [19]: from scipy.cluster.hierarchy import fcluster
k = 5
clusters = fcluster(Z, k, criterion='maxclust')
clusters
```

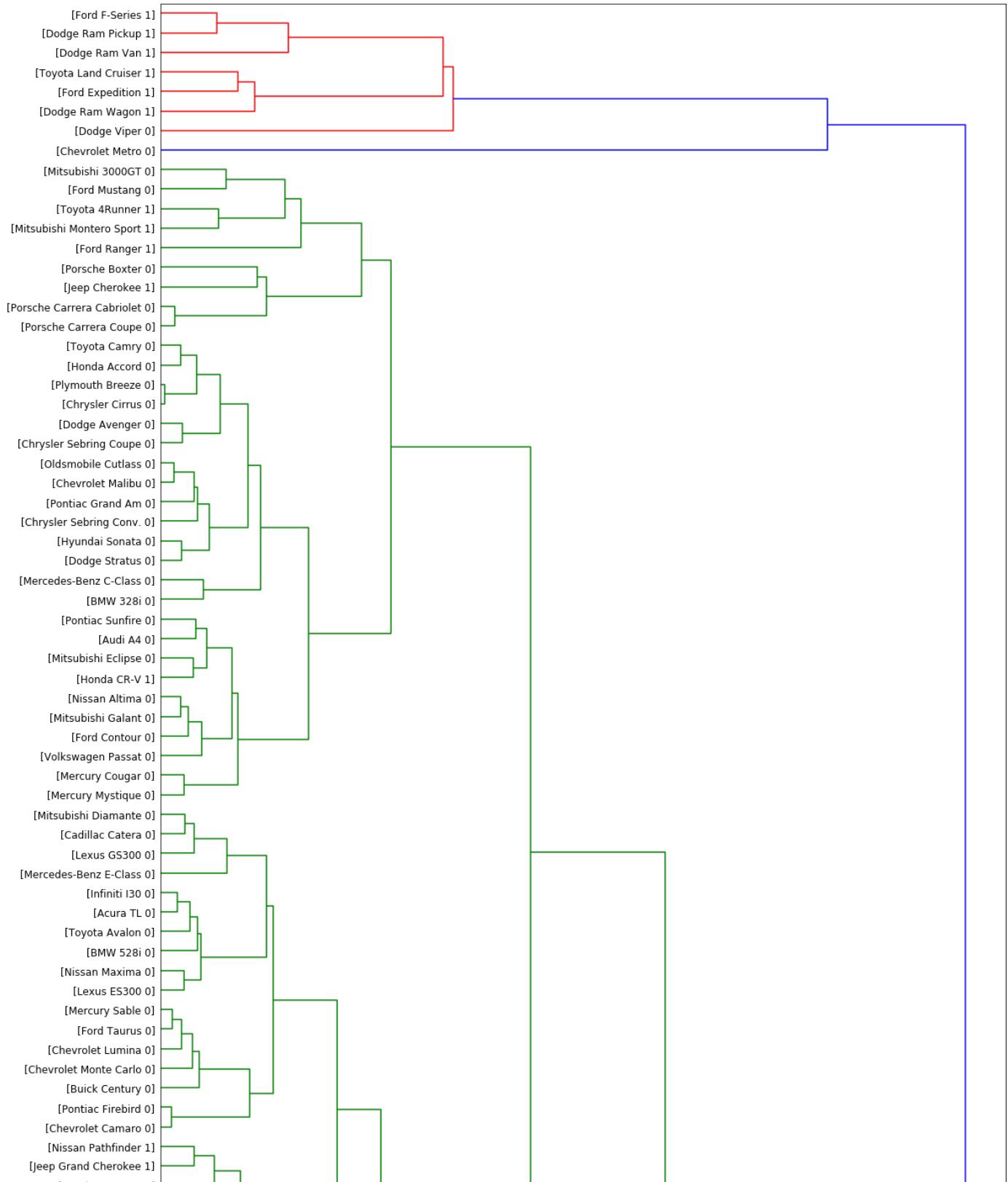
```
Out[19]: array([1, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3, 2, 2, 3, 1, 3, 3, 3, 3, 2, 1,
   5, 3, 3, 3, 3, 1, 3, 3, 4, 4, 4, 4, 4, 2, 3, 1, 3, 3, 3, 2, 3, 2, 1, 3,
   4, 3, 4, 1, 3, 3, 2, 1, 1, 3, 3, 1, 3, 3, 3, 2, 2, 1, 3, 3, 3, 2, 2, 1, 3,
   3, 3, 3, 2, 3, 3, 2, 3, 3, 3, 2, 2, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1,
   3, 2, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1,
   3, 4, 1, 1, 3, 1, 1], dtype=int32)
```

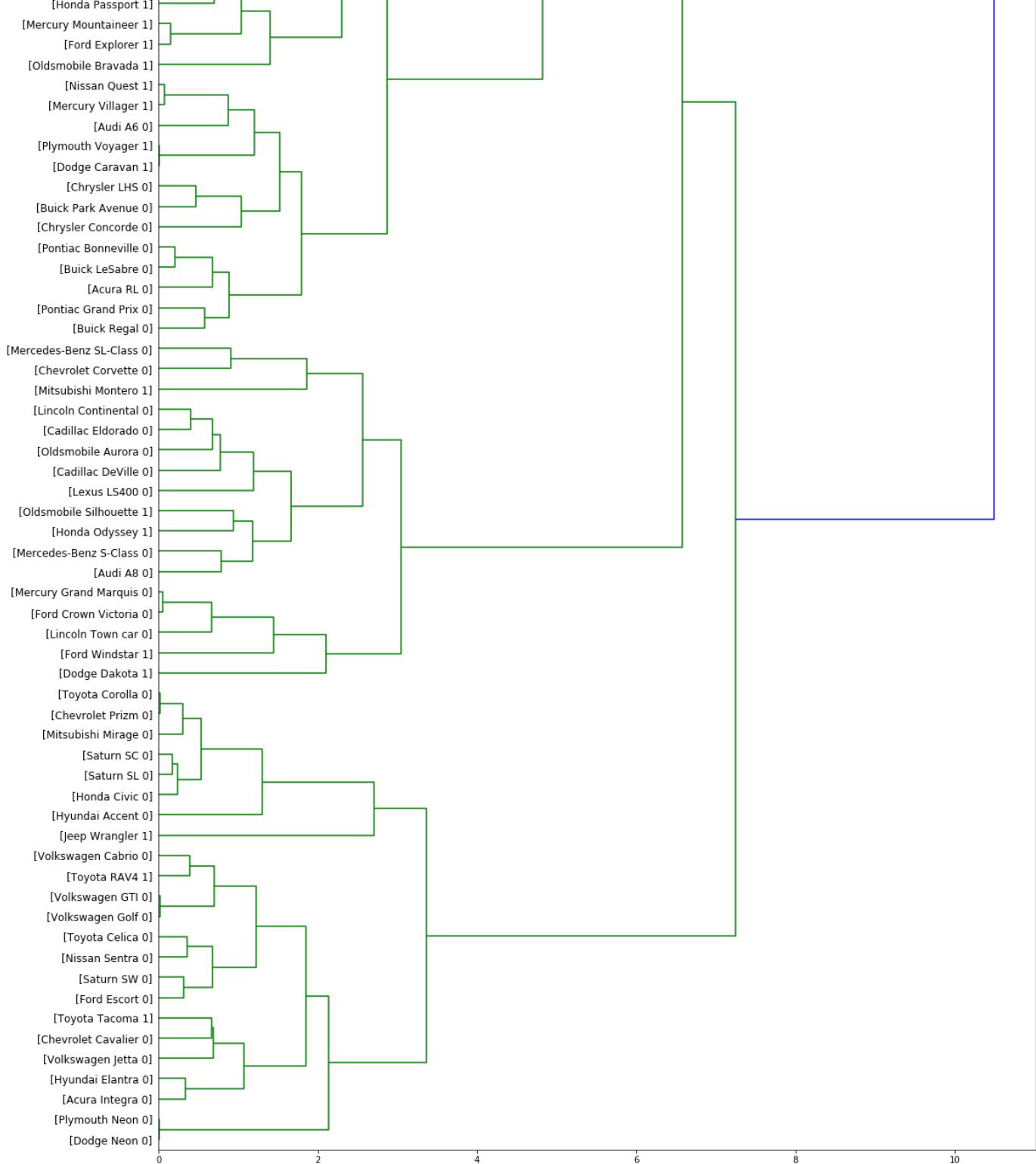
Now, plot the dendrogram:

```
In [20]: fig = pylab.figure(figsize=(18,50))
```

```
def llf(id):
    return '%s %s %s' % (pdf['manufact'][id], pdf['model'][id], int(float(pdf['type']) [
```

```
dendro = hierarchy.dendrogram(Z, leaf_label_func=llf, leaf_rotation=0, leaf_font_size =
```





Clustering using scikit-learn

Lets redo it again, but this time using scikit-learn package:

```
In [21]: dist_matrix = distance_matrix(feature_mtx, feature_mtx)
print(dist_matrix)
```

```
[ [0.          0.57777143  0.75455727 ... 0.28530295 0.24917241 0.18879995]
[0.57777143 0.          0.22798938 ... 0.36087756 0.66346677 0.62201282]
[0.75455727 0.22798938 0.          ... 0.51727787 0.81786095 0.77930119]
...
[0.28530295 0.36087756 0.51727787 ... 0.          0.41797928 0.35720492]
```

```
[0.24917241 0.66346677 0.81786095 ... 0.41797928 0. 0.15212198]
[0.18879995 0.62201282 0.77930119 ... 0.35720492 0.15212198 0. ]]
```

Now, we can use the 'AgglomerativeClustering' function from scikit-learn library to cluster the dataset. The AgglomerativeClustering performs a hierarchical clustering using a bottom up approach. The linkage criteria determines the metric used for the merge strategy:

- Ward minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.
- Maximum or complete linkage minimizes the maximum distance between observations of pairs of clusters.
- Average linkage minimizes the average of the distances between all observations of pairs of clusters.

```
In [22]: agglom = AgglomerativeClustering(n_clusters = 6, linkage = 'complete')
agglom.fit(feature_mtx)
agglom.labels_
```

```
J:\Anaconda3\lib\site-packages\sklearn\cluster\hierarchical.py:464: DeprecationWarning:
`np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

```
    children_ = out[:, :2].astype(np.int, copy=False)
```

```
Out[22]: array([1, 2, 2, 1, 2, 3, 1, 2, 2, 2, 2, 2, 3, 3, 2, 1, 1, 1, 2, 2, 2, 5, 1,
   4, 1, 1, 2, 1, 2, 1, 1, 1, 5, 0, 0, 0, 3, 2, 1, 2, 1, 2, 3, 2, 3,
   0, 3, 0, 1, 1, 1, 2, 3, 1, 1, 1, 2, 1, 1, 2, 2, 2, 3, 3, 3, 1, 1,
   1, 2, 1, 2, 2, 1, 1, 2, 3, 2, 3, 1, 2, 3, 5, 1, 1, 2, 3, 2, 1, 3,
   2, 3, 1, 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
   2, 0, 1, 1, 1, 1, 1], dtype=int64)
```

And, we can add a new field to our dataframe to show the cluster of each row:

```
In [23]: pdf['cluster_'] = agglom.labels_
pdf.head()
```

```
Out[23]:   manufact model  sales  resale  type  price  engine_s  horsepow  wheelbas  width  length  curb_wg
0      Acura  Integra  16.919  16.360  0.0  21.50       1.8     140.0     101.2    67.3   172.4    2.63
1      Acura       TL  39.384  19.875  0.0  28.40       3.2     225.0     108.1    70.3   192.9    3.51
2      Acura       RL   8.588  29.725  0.0  42.00       3.5     210.0     114.6    71.4   196.6    3.85
3      Audi       A4  20.397  22.255  0.0  23.99       1.8     150.0     102.6    68.2   178.0    2.99
4      Audi       A6  18.780  23.555  0.0  33.95       2.8     200.0     108.7    76.1   192.0    3.56
```

```
In [24]: import matplotlib.cm as cm
n_clusters = max(agglom.labels_)+1
colors = cm.rainbow(np.linspace(0, 1, n_clusters))
cluster_labels = list(range(0, n_clusters))

# Create a figure of size 6 inches by 4 inches.
plt.figure(figsize=(16,14))

for color, label in zip(colors, cluster_labels):
    subset = pdf[pdf.cluster_ == label]
    for i in subset.index:
```

```
plt.text(subset.horsepow[i], subset.mpg[i], str(subset['model'][i]), rotation=90)
plt.scatter(subset.horsepow, subset.mpg, s=subset.price*10, c=color, label='cluster')
# plt.scatter(subset.horsepow, subset.mpg)
plt.legend()
plt.title('Clusters')
plt.xlabel('horsepow')
plt.ylabel('mpg')
```

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

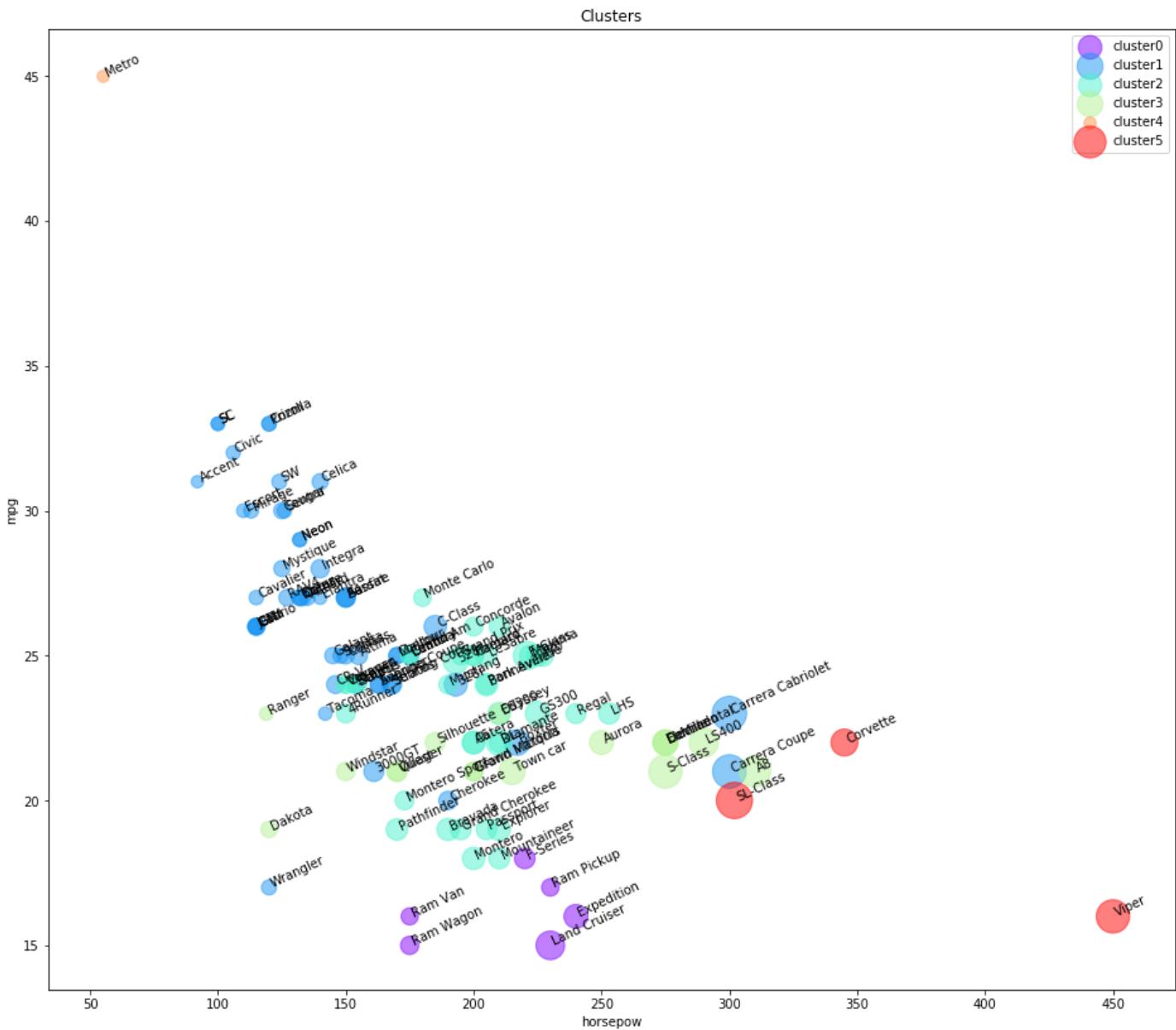
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

Out[24]:

Text(0, 0.5, 'mpg')



As you can see, we are seeing the distribution of each cluster using the scatter plot, but it is not very clear where is the centroid of each cluster. Moreover, there are 2 types of vehicles in our dataset, "truck" (value of 1 in the type column) and "car" (value of 1 in the type column). So, we use them to distinguish the classes, and summarize the cluster. First we count the number of cases in each group:

```
In [25]: pdf.groupby(['cluster_','type'])['cluster_'].count()
```

```
Out[25]: cluster_    type
0          1.0      6
1          0.0     47
           1.0      5
2          0.0     27
           1.0     11
3          0.0     10
           1.0      7
4          0.0      1
5          0.0      3
Name: cluster_, dtype: int64
```

Now we can look at the characteristics of each cluster:

```
In [26]: agg_cars = pdf.groupby(['cluster_','type'])[['horsepow','engine_s','mpg','price']].mean()
agg_cars
```

	horsepow	engine_s	mpg	price
--	----------	----------	-----	-------

Out[26]:

cluster_	type					
0	1.0	211.666667	4.483333	16.166667	29.024667	
1	0.0	146.531915	2.246809	27.021277	20.306128	
1	1.0	145.000000	2.580000	22.200000	17.009200	
2	0.0	203.111111	3.303704	24.214815	27.750593	
2	1.0	182.090909	3.345455	20.181818	26.265364	
3	0.0	256.500000	4.410000	21.500000	42.870400	
3	1.0	160.571429	3.071429	21.428571	21.527714	
4	0.0	55.000000	1.000000	45.000000	9.235000	
5	0.0	365.666667	6.233333	19.333333	66.010000	

It is obvious that we have 3 main clusters with the majority of vehicles in those.

Cars:

- Cluster 1: with almost high mpg, and low in horsepower.
- Cluster 2: with good mpg and horsepower, but higher price than average.
- Cluster 3: with low mpg, high horsepower, highest price.

Trucks:

- Cluster 1: with almost highest mpg among trucks, and lowest in horsepower and price.
- Cluster 2: with almost low mpg and medium horsepower, but higher price than average.
- Cluster 3: with good mpg and horsepower, low price.

Please notice that we did not use **type**, and **price** of cars in the clustering process, but Hierarchical clustering could forge the clusters and discriminate them with quite high accuracy.

In [30]:

```
plt.figure(figsize=(16,10))
for color, label in zip(colors, cluster_labels):
    subset = agg_cars.loc[(label),]
    for i in subset.index:
        plt.text(subset.loc[i][0]+5, subset.loc[i][2], 'type=' + str(int(i)) + ', price=' +
        plt.scatter(subset.horsepow, subset.mpg, s=subset.price*20, c=color, label='cluster')
plt.legend()
plt.title('Clusters')
plt.xlabel('horsepow')
plt.ylabel('mpg')
```

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

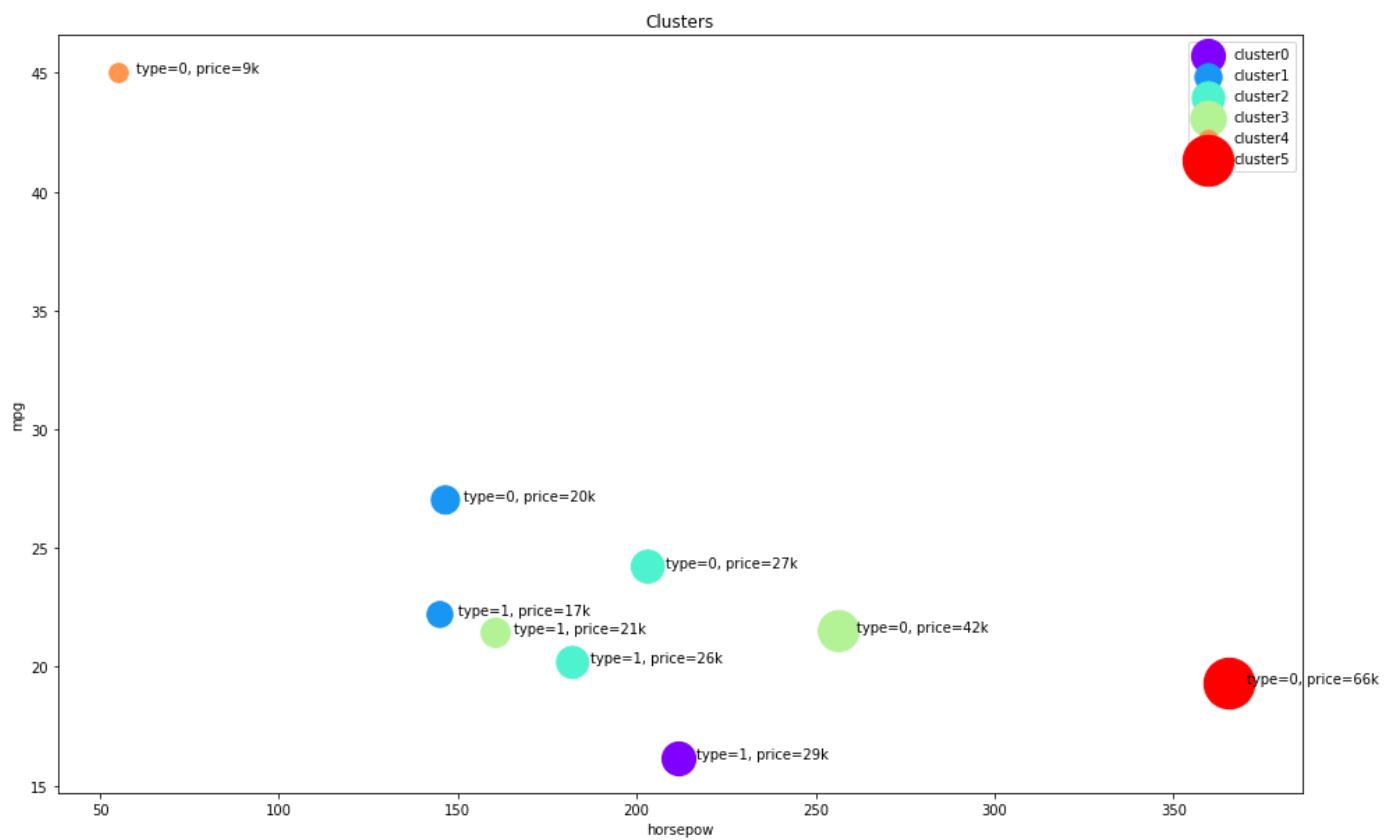
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA val

ue for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with 'x' & 'y'. Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all points.

Out[30]:



In []:

Часть 7

Decision Trees

Теоретические сведения

In this lab exercise, you will learn a popular machine learning algorithm, Decision Tree. You will use this classification algorithm to build a model from historical data of patients, and their response to different medications. Then you use the trained decision tree to predict the class of a unknown patient, or to find a proper drug for a new patient.

В этой лабораторной работе вы изучите популярный алгоритм машинного обучения — дерево решений. Вы будете использовать этот алгоритм классификации для построения модели на основе исторических данных пациентов и их реакции на различные лекарства. Затем вы используете обученное дерево решений, чтобы предсказать класс неизвестного пациента или найти подходящее лекарство для нового пациента.

Деревья решений являются наиболее важными элементами другого наиболее распространенного в настоящее время алгоритма — случайного леса. Они способны подгонять сложные наборы данных, позволяя пользователю увидеть, как было принято решение. Одно дерево решений не очень хороший предсказатель; однако, создавая их совокупность (лес) и собирая их прогнозы, можно получить один из самых мощных инструментов машинного обучения.

Они представляют собой иерархические древовидные структуры, состоящие из решающих правил вида «Если ..., то ...». Правила автоматически генерируются в процессе обучения на обучающем множестве и, поскольку они формулируются практически на естественном языке (например, «Если объём продаж более 1000 шт., то товар перспективный»), деревья решений как аналитические модели более вербализуемы и интерпретируемы, чем, скажем, нейронные сети.

Поскольку правила в деревьях решений получаются путём обобщения множества отдельных наблюдений (обучающих примеров), описывающих предметную область, то по аналогии с соответствующим методом логического вывода их называют индуктивными правилами, а сам процесс обучения — индукцией деревьев решений.

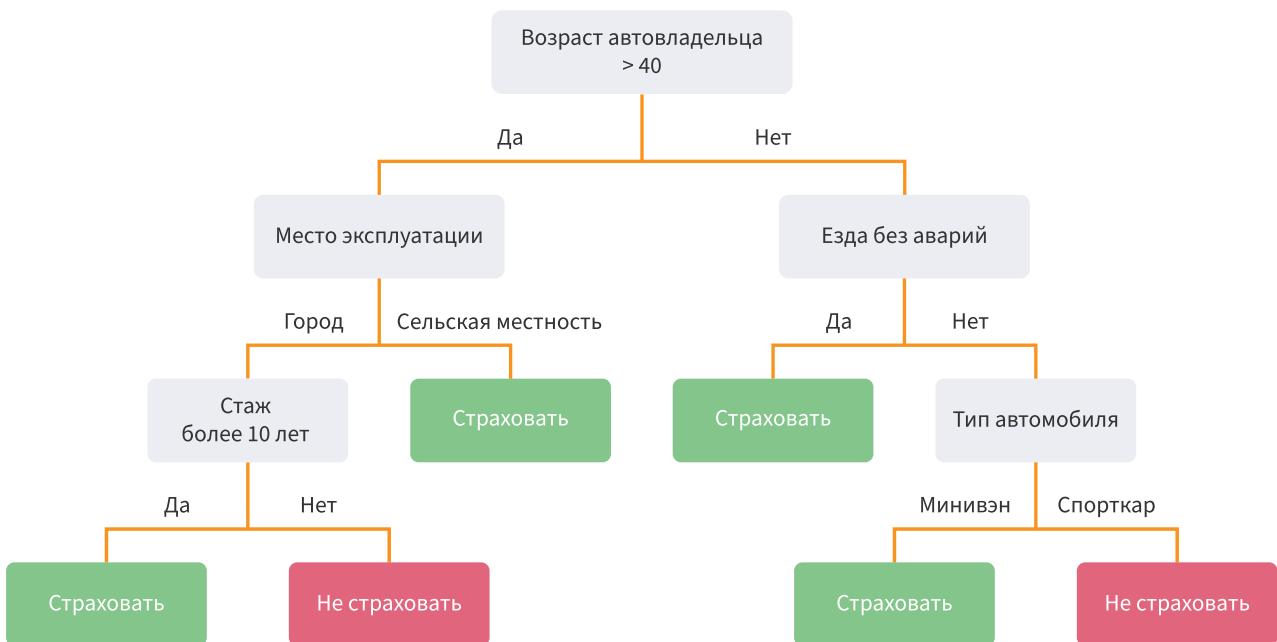
В обучающем множестве для примеров должно быть задано целевое значение, т.к. деревья решений являются моделями, строящимися на основе обучения с учителем. При этом, если целевая переменная дискретная (метка класса), то модель называют деревом классификации, а если непрерывная, то деревом регрессии.

Само дерево решений — это метод представления решающих правил в иерархической структуре, состоящей из элементов двух типов — узлов (node) и листьев (leaf). В узлах находятся

решающие правила и производится проверка соответствия примеров этому правилу по какому-либо атрибуту обучающего множества.

В простейшем случае, в результате проверки, множество примеров, попавших в узел, разбивается на два подмножества, в одно из которых попадают примеры, удовлетворяющие правило, а в другое — не удовлетворяющие. Затем к каждому подмножеству вновь применяется правило и процедура рекурсивно повторяется пока не будет достигнуто некоторое условие остановки алгоритма. В результате в последнем узле проверка и разбиение не производится и он объявляется листом. Лист определяет решение для каждого попавшего в него примера. Для дерева классификации — это класс, ассоциируемый с узлом, а для дерева регрессии — соответствующий листу модальный интервал целевой переменной.

В отличие от узла, в листе содержится не правило, а подмножество объектов, удовлетворяющих всем правилам ветви, которая заканчивается данным листом. Очевидно, чтобы попасть в лист, пример должен удовлетворять всем правилам, лежащим на пути к этому листу. Поскольку путь в дереве к каждому листу единственный, то и каждый пример может попасть только в один лист, что обеспечивает единственность решения.



Процесс построения деревьев решений заключается в последовательном, рекурсивном разбиении обучающего множества на подмножества с применением решающих правил в узлах. Процесс разбиения продолжается до тех пор, пока все узлы в конце всех ветвей не будут объявлены листьями. Объявление узла листом может произойти естественным образом (когда он будет содержать единственный объект, или объекты только одного класса), или по достижении некоторого условия остановки, задаваемого пользователем (например, минимально допустимое число примеров в узле или максимальная глубина дерева).

Алгоритмы построения деревьев решений относят к категории так называемых жадных алгоритмов. Жадными называются алгоритмы, которые допускают, что локально-оптимальные решения на каждом шаге (разбиения в узлах), приводят к оптимальному итоговому решению. В случае деревьев решений это означает, что если один раз был выбран атрибут, и по нему было произведено разбиение на подмножества, то алгоритм не может вернуться назад и выбрать

другой атрибут, который дал бы лучшее итоговое разбиение. Поэтому на этапе построения нельзя сказать обеспечит ли выбранный атрибут, в конечном итоге, оптимальное разбиение.

В основе большинства популярных алгоритмов обучения деревьев решений лежит принцип «разделяй и властвуй». Алгоритмически этот принцип реализуется следующим образом. Пусть задано обучающее множество S , содержащее n примеров, для каждого из которых задана метка класса C_i ($i=1..k$) и m атрибутов A_j ($j=1..m$), которые, как предполагается, определяют принадлежность объекта к тому или иному классу. Тогда возможны три случая:

1. Все примеры множества S имеют одинаковую метку класса C_i (т.е. все обучающие примеры относятся только к одному классу). Очевидно, что обучение в этом случае не имеет смысла, поскольку все примеры, предъявляемые модели, будут одного класса, который и «научится» распознавать модель. Само дерево решений в этом случае будет представлять собой лист, ассоциированный с классом C_i . Практическое использование такого дерева бессмысленно, поскольку любой новый объект оно будет относить только к этому классу.
2. Множество S вообще не содержит примеров, т.е. является пустым множеством. В этом случае для него тоже будет создан лист (применять правило, чтобы создать узел, к пустому множеству бессмысленно), класс которого будет выбран из другого множества (например, класс, который наиболее часто встречается в родительском множестве).
3. Множество S содержит обучающие примеры всех классов C_k . В этом случае требуется разбить множество S на подмножества, ассоциированные с классами. Для этого выбирается один из атрибутов A_j множества S который содержит два и более уникальных значения (a_1, a_2, \dots, a_p), где p — число уникальных значений признака. Затем множество S разбивается на p подмножеств (S_1, S_2, \dots, S_p), каждое из которых включает примеры, содержащие соответствующее значение атрибута. Затем выбирается следующий атрибут и разбиение повторяется. Это процедура будет рекурсивно повторяться до тех пор, пока все примеры в результирующих подмножествах не окажутся одного класса.

Очевидно, что при использовании данной методики, построение дерева решений будет происходить сверху вниз (от корневого узла к листьям).

Основные этапы построения В ходе построения дерева решений нужно решить несколько основных проблем, с каждой из которых связан соответствующий шаг процесса обучения:

1. Выбор атрибута, по которому будет производиться разбиение в данном узле (атрибута разбиения). При формировании правила для разбиения в очередном узле дерева необходимо выбрать атрибут, по которому это будет сделано. Общее правило для этого можно сформулировать следующим образом: выбранный атрибут должен разбить множество наблюдений в узле так, чтобы результирующие подмножества содержали примеры с одинаковыми метками класса, или были максимально приближены к этому, т.е. количество объектов из других классов («примесей») в каждом из этих множеств было как можно меньше. Для этого были выбраны различные критерии, наиболее популярными из которых стали теоретико-информационный и статистический.

1.1. Теоретико-информационный критерий

Критерий основан на понятиях теории информации, а именно — информационной энтропии. $H = -\sum((N_i/N) \log(N_i/N))$, где суммирование происходит по всем классам в исходном множестве, N_i — число примеров i -го класса, N — общее количество примеров в подмножестве. Энтропия может

рассматриваться как мера неоднородности подмножества по представленным в нём классам. Когда классы представлены в равных долях и неопределенность классификации наибольшая, энтропия также максимальна. Если все примеры в узле относятся к одному классу, т.е. $N=N_i$, логарифм от единицы обращает энтропию в ноль. Таким образом, лучшим атрибутом разбиения A_j будет тот, который обеспечит максимальное снижение энтропии результирующего подмножества относительно родительского. На практике, однако, говорят не об энтропии, а о величине, обратной ей, которая называется информацией. Тогда лучшим атрибутом разбиения будет тот, который обеспечит максимальный прирост информации результирующего узла относительно исходного: $\text{Gain}(A) = \text{Info}(S) - \text{Info}(S_A)$, где $\text{Info}(S)$ – информация, связанная с подмножеством S до разбиения; $\text{Info}(S_A)$ – информация, связанная с подмножеством, полученными при разбиении по атрибуту A . Таким образом, задача выбора атрибута разбиения в узле заключается в максимизации величины $\text{Gain}(A)$, называемой приростом информации (от англ. gain – прирост, увеличение). Поэтому сам теоретико-информационный подход известен как критерий прироста информации.

1.2. Статистический подход

В основе статистического подхода лежит использование индекса Джини (назван в честь итальянского статистика и экономиста Коррадо Джини). Статистический смысл данного показателя в том, что он показывает – насколько часто случайно выбранный пример обучающего множества будет распознан неправильно, при условии, что целевые значения в этом множестве были взяты из определённого статистического распределения. Таким образом индекс Джини фактически показывает расстояние между двумя распределениями – распределением целевых значений, и распределением предсказаний модели. Очевидно, что чем меньше данное расстояние, тем лучше работает модель. Индекс Джини может быть рассчитан по формуле: $\text{Gini}(Q) = 1 - \sum((p_i)^2)$, где Q – результирующее множество, n – число классов в нём, p_i – вероятность i -го класса (выраженная как относительная частота примеров соответствующего класса). Очевидно, что данный показатель меняется от 0 до 1. При этом он равен 0, если все примеры Q относятся к одному классу, и равен 1, когда классы представлены в равных пропорциях и равновероятны. Тогда лучшим будет то разбиение, для которого значение индекса Джини будут минимальным.

1. Выбор критерия остановки обучения. Теоретически, алгоритм обучения дерева решений будет работать до тех пор, пока в результате не будут получены абсолютно «чистые» подмножества, в каждом из которых будут примеры одного класса. Правда, возможно при этом будет построено дерево, в котором для каждого примера будет создан отдельный лист. Очевидно, что такое дерево окажется бесполезным, поскольку оно будет переобученным – каждому примеру будет соответствовать свой уникальный путь в дереве, а следовательно, и набор правил, актуальный только для данного примера.

Переобучение в случае дерева решений ведёт к тем же последствиям, что и для нейронной сети – точное распознавание примеров, участвующих в обучении и полная несостоительность на новых данных. Кроме этого, переобученные деревья имеют очень сложную структуру, и поэтому их сложно интерпретировать.

Очевидным решением проблемы является принудительная остановка построения дерева, пока оно не стало переобученным. Для этого разработаны следующие подходы.

- Ранняя остановка – алгоритм будет остановлен, как только будет достигнуто заданное значение некоторого критерия, например процентной доли правильно распознанных

примеров. Единственным преимуществом подхода является снижение времени обучения. Главным недостатком является то, что ранняя остановка всегда делается в ущерб точности дерева, поэтому многие авторы рекомендуют отдавать предпочтение отсечению ветвей.

- Ограничение глубины дерева — задание максимального числа разбиений в ветвях, по достижении которого обучение останавливается. Данный метод также ведёт к снижению точности дерева.
- Задание минимально допустимого числа примеров в узле — запретить алгоритму создавать узлы с числом примеров меньше заданного (например, 5). Это позволит избежать создания тривиальных разбиений и, соответственно, малозначимых правил.

Все перечисленные подходы являются эвристическими, т.е. не гарантируют лучшего результата или вообще работают только в каких-то частных случаях. Поэтому к их использованию следует подходить с осторожностью. Каких-либо обоснованных рекомендаций по тому, какой метод лучше работает, в настоящее время тоже не существует. Поэтому аналитикам приходится использовать метод проб и ошибок.

1. Выбор метода отсечения ветвей (упрощения). Как было отмечено выше, если «рост» дерева не ограничить, то в результате будет построено сложное дерево с большим числом узлов и листьев. Как следствие оно будет трудно интерпретируемым. В то же время решающие правила в таких деревьях, создающие узлы, в которые попадают два-три примера, оказываются малозначимыми с практической точки зрения.

Гораздо предпочтительнее иметь дерево, состоящее из малого количества узлов, которым бы соответствовало большое число примеров из обучающей выборки. Поэтому представляет интерес подход, альтернативный ранней остановке — построить все возможные деревья и выбрать то из них, которое при разумной глубине обеспечивает приемлемый уровень ошибки распознавания, т.е. найти наиболее выгодный баланс между сложностью и точностью дерева. К сожалению, это задача относится к классу NP-полных задач, что было показано Л. Хайфилем (L. Hyafil) и Р. Ривестом (R. Rivest), и, как известно, этот класс задач не имеет эффективных методов решения. Альтернативным подходом является так называемое отсечение ветвей (pruning). Он содержит следующие шаги:

- Построить полное дерево (чтобы все листья содержали примеры одного класса).
 - Определить два показателя: относительную точность модели — отношение числа правильно распознанных примеров к общему числу примеров, и абсолютную ошибку — число неправильно классифицированных примеров.
 - Удалить из дерева листья и узлы, отсечение которых не приведёт к значимому уменьшению точности модели или увеличению ошибки. Отсечение ветвей, очевидно, производится в направлении, противоположном направлению роста дерева, т.е. снизу вверх, путём последовательного преобразования узлов в листья. Преимуществом отсечения ветвей по сравнению с ранней остановкой является возможность поиска оптимального соотношения между точностью и понятностью дерева. Недостатком является большее время обучения из-за необходимости сначала построить полное дерево.
1. Извлечение правил. Иногда даже упрощённое дерево решений все ещё является слишком сложным для визуального восприятия и интерпретации. В этом случае может оказаться полезным извлечь из дерева решающие правила и организовать их в наборы, описывающие классы. Для извлечения правил нужно отследить все пути от корневого узла к листьям дерева. Каждый такой путь даст правило, состоящее из множества условий,

представляющих собой проверку в каждом узле пути. Визуализация сложных деревьев решений в виде решающих правил вместо иерархической структуры из узлов и листьев может оказаться более удобной для визуального восприятия.

Программа работы

1. About the dataset
2. Downloading the Data
3. Pre-processing
4. Setting up the Decision Tree
5. Modeling
6. Prediction
7. Evaluation
8. Visualization

Import the Following Libraries:

- **numpy (as np)**
- **pandas**
- **DecisionTreeClassifier from sklearn.tree**

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
```

About the dataset

Imagine that you are a medical researcher compiling data for a study. You have collected data about a set of patients, all of whom suffered from the same illness. During their course of treatment, each patient responded to one of 5 medications, Drug A, Drug B, Drug c, Drug x and y.

Part of your job is to build a model to find out which drug might be appropriate for a future patient with the same illness. The feature sets of this dataset are Age, Sex, Blood Pressure, and Cholesterol of patients, and the target is the drug that each patient responded to.

It is a sample of binary classifier, and you can use the training part of the dataset to build a decision tree, and then use it to predict the class of a unknown patient, or to prescribe it to a new patient.

Представьте, что вы медицинский исследователь, собирающий данные для исследования. Вы собрали данные о группе пациентов, все из которых страдали одним и тем же заболеванием. Во время курса лечения каждый пациент реагировал на одно из 5 лекарств: препарат А, препарат В, препарат С, препарат Х и У.

Часть вашей работы состоит в том, чтобы построить модель, чтобы выяснить, какое лекарство может быть подходящим для будущего пациента с таким же заболеванием. Наборами признаков этого набора данных являются возраст, пол, кровяное давление и уровень холестерина пациентов, а целью является препарат, на который отреагировал каждый пациент.

Это образец бинарного классификатора, и вы можете использовать обучающую часть набора данных для построения дерева решений, а затем использовать его для прогнозирования класса неизвестного пациента или для назначения его новому пациенту.

Read data using pandas dataframe:

```
In [2]: my_data = pd.read_csv("drug200.csv", delimiter=",")  
my_data[0:5]
```

```
Out[2]:
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

Practice

What is the size of data?

```
In [3]: # write your code here  
my_data.shape  
my_data.Drug.value_counts()
```

```
Out[3]: drugY    91  
drugX    54  
drugA    23  
drugB    16  
drugC    16  
Name: Drug, dtype: int64
```

Pre-processing

Using **my_data** as the Drug.csv data read by pandas, declare the following variables:

- **X** as the **Feature Matrix** (data of my_data)
- **y** as the **response vector (target)**

Remove the column containing the target name since it doesn't contain numeric values.

```
In [4]: X = my_data[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values  
X[0:5]
```

```
Out[4]: array([[23, 'F', 'HIGH', 'HIGH', 25.355],  
               [47, 'M', 'LOW', 'HIGH', 13.093],  
               [47, 'M', 'LOW', 'HIGH', 10.11399999999999],  
               [28, 'F', 'NORMAL', 'HIGH', 7.79799999999999],  
               [61, 'F', 'LOW', 'HIGH', 18.043]], dtype=object)
```

As you may figure out, some features in this dataset are categorical such as **Sex** or **BP**. Unfortunately, Sklearn Decision Trees do not handle categorical variables. But still we can convert these features to numerical values. **pandas.get_dummies()** Convert categorical variable into dummy/indicator variables.

```
In [5]: from sklearn import preprocessing
le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F','M'])
X[:,1] = le_sex.transform(X[:,1])

le_BP = preprocessing.LabelEncoder()
le_BP.fit(['LOW', 'NORMAL', 'HIGH'])
X[:,2] = le_BP.transform(X[:,2])

le_Chol = preprocessing.LabelEncoder()
le_Chol.fit(['NORMAL', 'HIGH'])
X[:,3] = le_Chol.transform(X[:,3])

X[0:5]
```

```
Out[5]: array([[23, 0, 0, 0, 25.355],
   [47, 1, 1, 0, 13.093],
   [47, 1, 1, 0, 10.11399999999999],
   [28, 0, 2, 0, 7.79799999999999],
   [61, 0, 1, 0, 18.043]], dtype=object)
```

Now we can fill the target variable.

```
In [6]: y = my_data["Drug"]
y[0:5]
```

```
Out[6]: 0    drugY
1    drugC
2    drugC
3    drugX
4    drugY
Name: Drug, dtype: object
```

Setting up the Decision Tree

We will be using **train/test split** on our **decision tree**. Let's import **train_test_split** from **sklearn.cross_validation**.

```
In [7]: from sklearn.model_selection import train_test_split
```

Now **train_test_split** will return 4 different parameters. We will name them:
X_trainset, X_testset, y_trainset, y_testset

The **train_test_split** will need the parameters:
X, y, test_size=0.3, and random_state=3.

The **X** and **y** are the arrays required before the split, the **test_size** represents the ratio of the testing dataset, and the **random_state** ensures that we obtain the same splits.

```
In [8]: x_trainset, x_testset, y_trainset, y_testset = train_test_split(X, y, test_size=0.3, ran
```

Practice

Print the shape of X_trainset and y_trainset. Ensure that the dimensions match

In [9]:

```
# your code  
print(X_trainset.shape)  
print(y_trainset.shape)
```

(140, 5)
(140,)

Print the shape of X_testset and y_testset. Ensure that the dimensions match

In [10]:

```
# your code  
  
print(X_testset.shape)  
print(y_testset.shape)
```

(60, 5)
(60,)

Modeling

We will first create an instance of the **DecisionTreeClassifier** called **drugTree**.

Inside of the classifier, specify *criterion="entropy"* so we can see the information gain of each node.

In [11]:

```
drugTree = DecisionTreeClassifier(criterion="entropy", max_depth = 4)  
drugTree # it shows the default parameters
```

Out[11]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,  
                      max_features=None, max_leaf_nodes=None,  
                      min_impurity_decrease=0.0, min_impurity_split=None,  
                      min_samples_leaf=1, min_samples_split=2,  
                      min_weight_fraction_leaf=0.0, presort=False,  
                      random_state=None, splitter='best')
```

Next, we will fit the data with the training feature matrix **X_trainset** and training response vector **y_trainset**

In [12]:

```
drugTree.fit(X_trainset,y_trainset)
```

Out[12]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,  
                      max_features=None, max_leaf_nodes=None,  
                      min_impurity_decrease=0.0, min_impurity_split=None,  
                      min_samples_leaf=1, min_samples_split=2,  
                      min_weight_fraction_leaf=0.0, presort=False,  
                      random_state=None, splitter='best')
```

Prediction

Let's make some **predictions** on the testing dataset and store it into a variable called **predTree**.

In [13]:

```
predTree = drugTree.predict(X_testset)
```

You can print out **predTree** and **y_testset** if you want to visually compare the prediction to the actual values.

```
In [14]: print (predTree [0:5])
print (y_testset [0:5])

['drugY' 'drugX' 'drugX' 'drugX' 'drugX']
40    drugY
51    drugX
139   drugX
197   drugX
170   drugX
Name: Drug, dtype: object
```

Evaluation

Next, let's import **metrics** from sklearn and check the accuracy of our model.

```
In [15]: from sklearn import metrics
import matplotlib.pyplot as plt
print("DecisionTrees's Accuracy: ", metrics.accuracy_score(y_testset, predTree))

DecisionTrees's Accuracy:  0.9833333333333333
```

Accuracy classification score computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in `y_true`.

In multilabel classification, the function returns the subset accuracy. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

Visualization

Lets visualize the tree

```
In [16]: # Notice: You might need to uncomment and install the pydotplus and graphviz libraries if
# conda install -c conda-forge pydotplus -y
# conda install -c conda-forge python-graphviz -y

Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.

Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

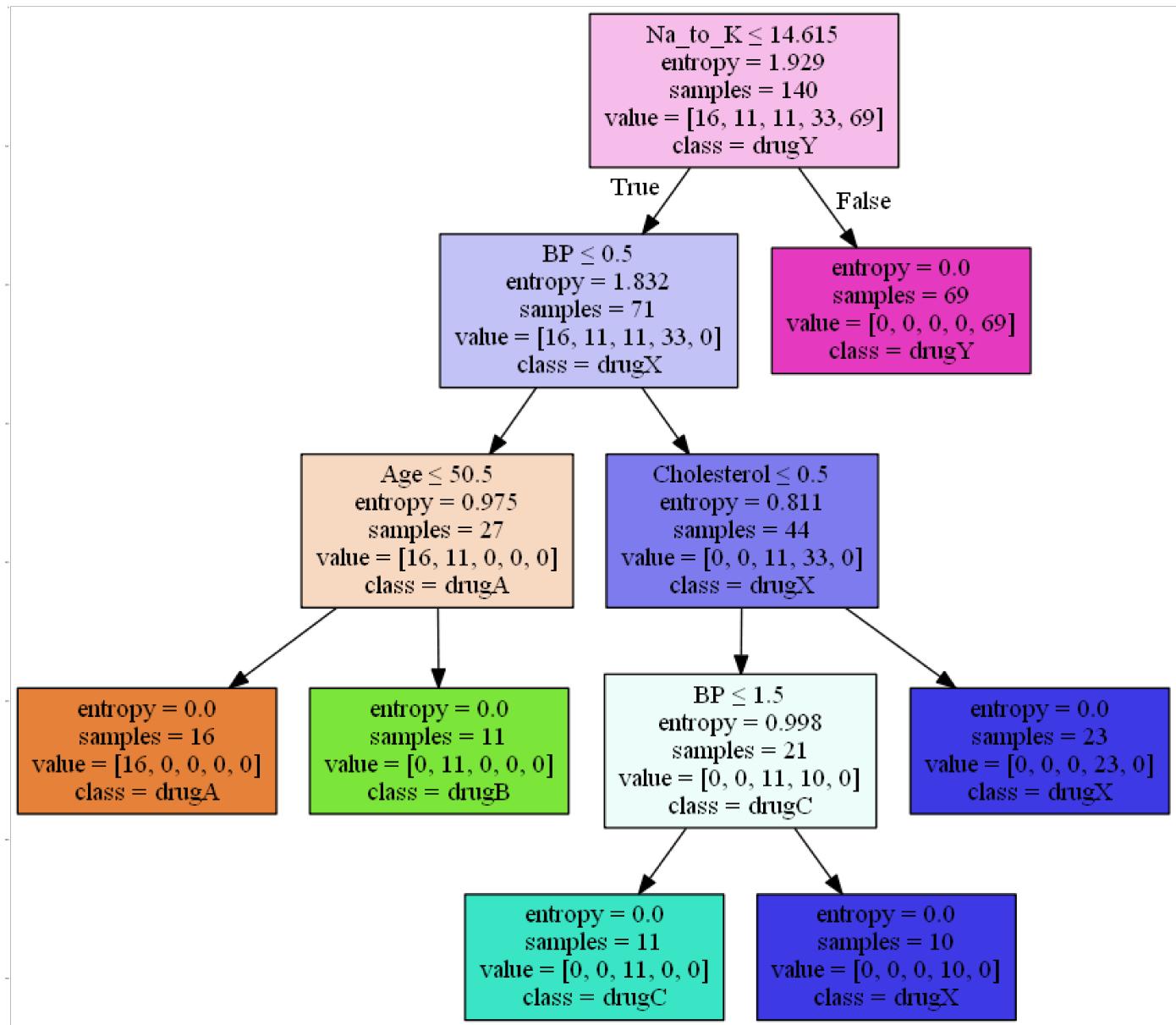
# All requested packages already installed.
```

```
In [17]: from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
%matplotlib inline
```

J:\Anaconda3\lib\site-packages\sklearn\externals\six.py:31: DeprecationWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (<https://pypi.org/project/six/>).
"(<https://pypi.org/project/six/>).", DeprecationWarning)

```
In [18]: dot_data = StringIO()
filename = "drugtree.png"
featureNames = my_data.columns[0:5]
targetNames = my_data["Drug"].unique().tolist()
out=tree.export_graphviz(drugTree, feature_names=featureNames, out_file=dot_data, class_n
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png(filename)
img = mpimg.imread(filename)
plt.figure(figsize=(100, 200))
plt.imshow(img, interpolation='nearest')
```

Out[18]: <matplotlib.image.AxesImage at 0x1370ac49908>



In []:

Часть 8

SVM (Support Vector Machines)

Теоретические сведения

In this notebook, you will use SVM (Support Vector Machines) to build and train a model using human cell records, and classify cells to whether the samples are benign or malignant.

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data is transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

В этой лабораторной работе вы будете использовать SVM (машины опорных векторов) для построения и обучения модели с использованием записей о клетках человека и классификации клеток по доброкачественности или злокачественности образцов.

SVM работает, отображая данные в многомерное пространство признаков, чтобы точки данных можно было классифицировать, даже если данные не могут быть линейно разделены иным образом. Находится разделитель между категориями, затем данные преобразуются таким образом, чтобы разделитель можно было изобразить в виде гиперплоскости. После этого характеристики новых данных можно использовать для прогнозирования группы, к которой должна принадлежать новая запись.

Программа работы

1. Load the Cancer data
2. Modeling
3. Evaluation
4. Practice

In [1]:

```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
%matplotlib inline
import matplotlib.pyplot as plt
```

Load the Cancer data

The example is based on a dataset that is publicly available from the UCI Machine Learning Repository (Asuncion and Newman, 2007)[<http://mlearn.ics.uci.edu/MLRepository.html>]. The dataset consists of several hundred human cell sample records, each of which contains the values of a set of cell characteristics. The fields in each record are:

Field name	Description
ID	Clump thickness
Clump	Clump thickness
UnifSize	Uniformity of cell size
UnifShape	Uniformity of cell shape
MargAdh	Marginal adhesion
SingEpiSize	Single epithelial cell size
BareNuc	Bare nuclei
BlandChrom	Bland chromatin
NormNucl	Normal nucleoli
Mit	Mitoses
Class	Benign or malignant

For the purposes of this example, we're using a dataset that has a relatively small number of predictors in each record. To download the data, we will use `!wget` to download it from IBM Object Storage.

Did you know? When it comes to Machine Learning, you will likely be working with large datasets. As a business, where can you host your data? IBM is offering a unique opportunity for businesses, with 10 Tb of IBM Cloud Object Storage: [Sign up now for free](#)

In [2]: #Click here and press Shift+Enter

```
#!wget -O cell_samples.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-
```

Load Data From CSV File

In [3]:

```
cell_df = pd.read_csv("cell_samples.csv")
cell_df.head()
```

Out[3]:

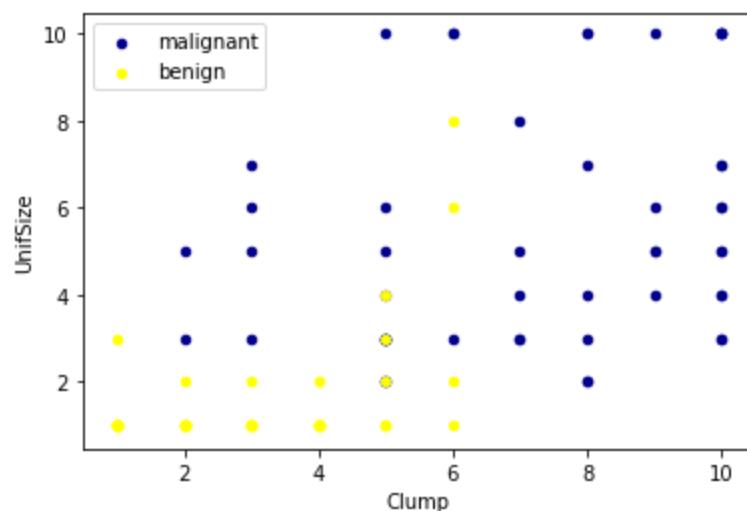
	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	C
0	1000025	5	1	1	1	2	1	3	1	1	
1	1002945	5	4	4	5	7	10	3	2	1	
2	1015425	3	1	1	1	2	2	3	1	1	
3	1016277	6	8	8	1	3	4	3	7	1	
4	1017023	4	1	1	3	2	1	3	1	1	

The ID field contains the patient identifiers. The characteristics of the cell samples from each patient are contained in fields Clump to Mit. The values are graded from 1 to 10, with 1 being the closest to benign.

The Class field contains the diagnosis, as confirmed by separate medical procedures, as to whether the samples are benign (value = 2) or malignant (value = 4).

Lets look at the distribution of the classes based on Clump thickness and Uniformity of cell size:

```
In [4]: ax = cell_df[cell_df['Class'] == 4][0:50].plot(kind='scatter', x='Clump', y='UnifSize', color='blue')
cell_df[cell_df['Class'] == 2][0:50].plot(kind='scatter', x='Clump', y='UnifSize', color='yellow')
plt.show()
```



Data pre-processing and selection

Lets first look at columns data types:

```
In [5]: cell_df.dtypes
```

```
Out[5]: ID          int64
Clump       int64
UnifSize    int64
UnifShape   int64
MargAdh     int64
SingEpiSize int64
BareNuc     object
BlandChrom  int64
NormNucl    int64
Mit         int64
Class        int64
dtype: object
```

It looks like the **BareNuc** column includes some values that are not numerical. We can drop those rows:

```
In [6]: cell_df = cell_df[pd.to_numeric(cell_df['BareNuc'], errors='coerce').notnull()]
cell_df['BareNuc'] = cell_df['BareNuc'].astype('int')
cell_df.dtypes
```

```
Out[6]: ID          int64
Clump       int64
UnifSize    int64
UnifShape   int64
MargAdh     int64
SingEpiSize int64
```

```
BareNuc      int32
BlandChrom   int64
NormNucl     int64
Mit          int64
Class        int64
dtype: object
```

```
In [7]: feature_df = cell_df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareN
X = np.asarray(feature_df)
X[0:5]
```

```
Out[7]: array([[ 5,  1,  1,  1,  2,  1,  3,  1,  1],
   [ 5,  4,  4,  5,  7, 10,  3,  2,  1],
   [ 3,  1,  1,  1,  2,  2,  3,  1,  1],
   [ 6,  8,  8,  1,  3,  4,  3,  7,  1],
   [ 4,  1,  1,  3,  2,  1,  3,  1,  1]], dtype=int64)
```

We want the model to predict the value of Class (that is, benign (=2) or malignant (=4)). As this field can have one of only two possible values, we need to change its measurement level to reflect this.

```
In [8]: cell_df['Class'] = cell_df['Class'].astype('int')
y = np.asarray(cell_df['Class'])
y[0:5]
```

```
Out[8]: array([2, 2, 2, 2, 2])
```

Train/Test dataset

Okay, we split our dataset into train and test set:

```
In [9]: X_train, X_test, y_train, y_test = train_test_split( x, y, test_size=0.2, random_state=4
print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)

Train set: (546, 9) (546,)
Test set: (137, 9) (137,)
```

Modeling (SVM with Scikit-learn)

The SVM algorithm offers a choice of kernel functions for performing its processing. Basically, mapping data into a higher dimensional space is called kernelling. The mathematical function used for the transformation is known as the kernel function, and can be of different types, such as:

- 1.Linear
- 2.Polynomial
- 3.Radial basis function (RBF)
- 4.Sigmoid

Each of these functions has its characteristics, its pros and cons, and its equation, but as there's no easy way of knowing which function performs best with any given dataset, we usually choose different functions in turn and compare the results. Let's just use the default, RBF (Radial Basis Function) for this lab.

```
In [10]: from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

```
J:\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: The default value  
of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscal  
ed features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.  
    "avoid this warning.", FutureWarning)
```

```
Out[10]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,  
             decision_function_shape='ovr', degree=3, gamma='auto_deprecated',  
             kernel='rbf', max_iter=-1, probability=False, random_state=None,  
             shrinking=True, tol=0.001, verbose=False)
```

After being fitted, the model can then be used to predict new values:

```
In [11]: yhat = clf.predict(X_test)  
yhat [0:5]
```

```
Out[11]: array([2, 4, 2, 4, 2])
```

Evaluation

```
In [12]: from sklearn.metrics import classification_report, confusion_matrix  
import itertools
```

```
In [13]: def plot_confusion_matrix(cm, classes,  
                           normalize=False,  
                           title='Confusion matrix',  
                           cmap=plt.cm.Blues):  
    """  
    This function prints and plots the confusion matrix.  
    Normalization can be applied by setting `normalize=True`.  
    """  
    if normalize:  
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
        print("Normalized confusion matrix")  
    else:  
        print('Confusion matrix, without normalization')  
  
    print(cm)  
  
    plt.imshow(cm, interpolation='nearest', cmap=cmap)  
    plt.title(title)  
    plt.colorbar()  
    tick_marks = np.arange(len(classes))  
    plt.xticks(tick_marks, classes, rotation=45)  
    plt.yticks(tick_marks, classes)  
  
    fmt = '.2f' if normalize else 'd'  
    thresh = cm.max() / 2.  
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
        plt.text(j, i, format(cm[i, j], fmt),  
                 horizontalalignment="center",  
                 color="white" if cm[i, j] > thresh else "black")  
  
    plt.tight_layout()  
    plt.ylabel('True label')  
    plt.xlabel('Predicted label')
```

```
In [14]: # Compute confusion matrix  
cnf_matrix = confusion_matrix(y_test, yhat, labels=[2, 4])  
np.set_printoptions(precision=2)  
  
print (classification_report(y_test, yhat))  
  
# Plot non-normalized confusion matrix
```

```

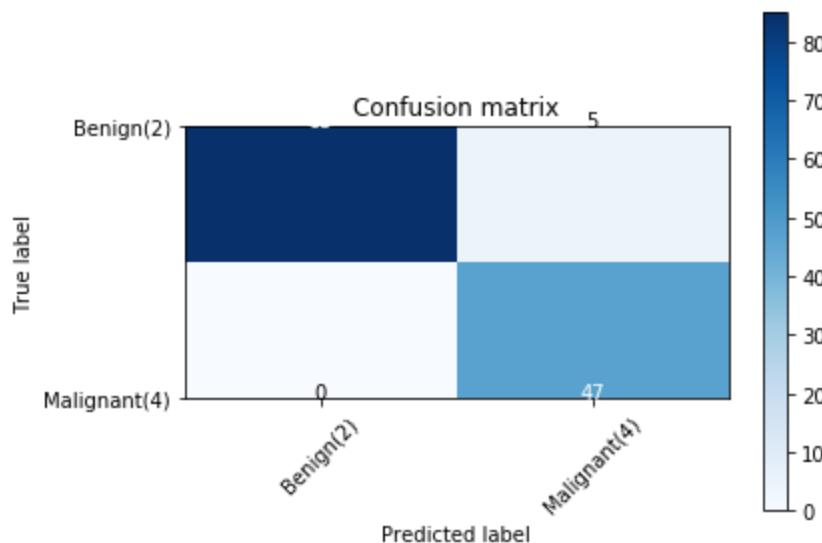
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['Benign(2)', 'Malignant(4)'], normalize=False)

precision    recall   f1-score   support
2           1.00    0.94    0.97     90
4           0.90    1.00    0.95     47

accuracy                           0.96    137
macro avg       0.95    0.97    0.96    137
weighted avg    0.97    0.96    0.96    137

Confusion matrix, without normalization
[[85  5]
 [ 0 47]]

```



You can also easily use the **f1_score** from sklearn library:

```
In [15]: from sklearn.metrics import f1_score
f1_score(y_test, yhat, average='weighted')
```

```
Out[15]: 0.9639038982104676
```

Lets try jaccard index for accuracy:

```
In [16]: from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

```
J:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:635: DeprecationWarning: jaccard_similarity_score has been deprecated and replaced with jaccard_score. It will be removed in version 0.23. This implementation has surprising behavior for binary and multiclass classification tasks.
'and multiclass classification tasks.', DeprecationWarning)
0.9635036496350365
```

Practice

Can you rebuild the model, but this time with a **linear** kernel? You can use **kernel='linear'** option, when you define the svm. How the accuracy changes with the new kernel function?

```
In [17]: # write your code here
```

Double-click **here** for the solution.

Практическая работа №4

Сохранение результатов измерений в базы данных

Цель работы: изучить средства взаимодействия с простейшими базами данных, в т.ч. в виде отдельных файлов.

Работа предусматривает три части, каждая из которых рассматривает отдельный алгоритм обработки данных.

Часть 1

Основы Anaconda и Jupyter Notebook

Одним из наиболее распространенных средств создания AI-систем является язык программирования Python. Стоит учитывать наличие значительного количества сред программирования для данного ЯП. Наиболее распространенные:

- Python Idle – стандартная среда программирования, не требует значительных ресурсов, используется для отладки программ «на месте» (например, в различных micro-PC таких как Raspberry Pi, Orange Pi и т.д.);

- Spyder – более функциональная среда по сравнению с Python Idle, используется для разработки и первично отладки программ на языке Python;

- Jupyter Notebook – среда разработки, апробации и первичной отладки алгоритмов интеллектуальной обработки данных (например, апробации различных вариантов искусственных НС для последующего встраивания их в общую программу).

Для использования всех указанных выше инструментов необходимо установить дистрибутив языков программирования Python и R, включающий набор популярных свободных библиотек, объединённых проблематиками науки о данных и машинного обучения (<https://www.anaconda.com/download/#windows> или google -> anaconda download).

Процесс установки достаточно прост и не отличается от установки любой другой программы. После установки в списке программ появится Anaconda Navigator (рис.).

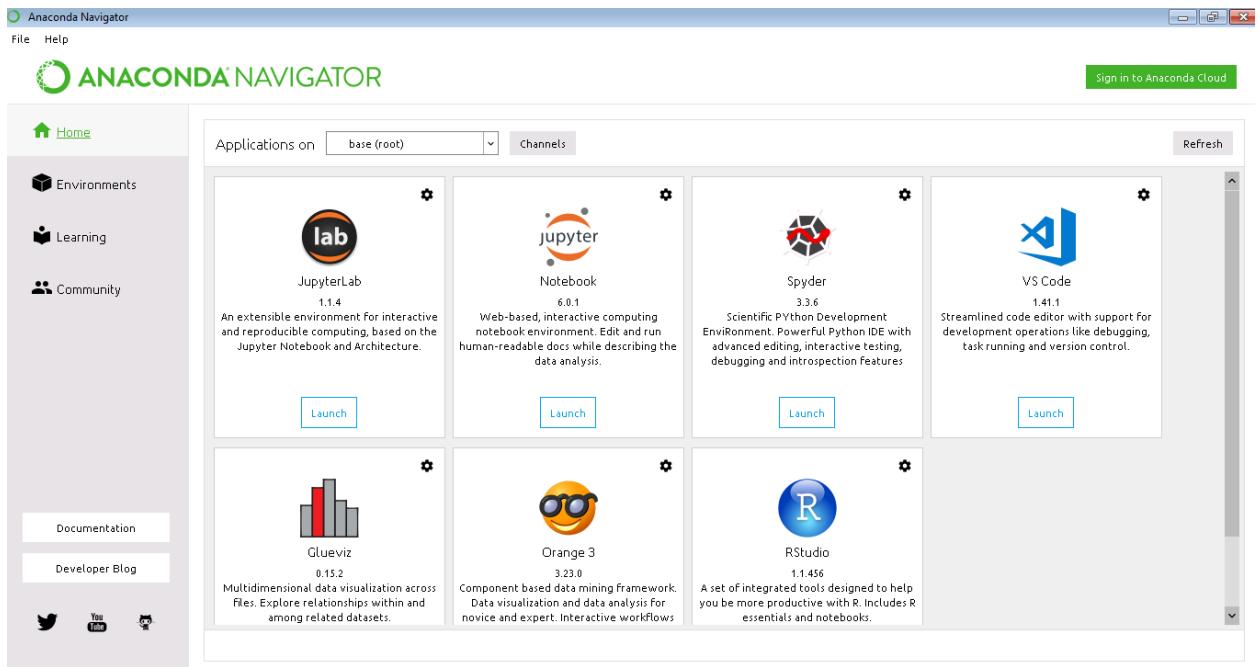


Рисунок – Anaconda Navigator

Запускаем jupyter Notebook (“Launch”) – интерфейсом программирования будет Ваш браузер (рис.).



Рисунок – Jupyter Notebook (директории зависят от Вашего компьютера)

1. В правом верхнем углу нажать – “New” → “Folder”. В списке появится “Untitled Folder” (папка без названия).
2. Поставить галочку слева от имени “Untitled Folder”.
3. В левом верхнем углу появится кнопка “Rename” (переименовать) → вводим имя каталога (work1).
4. Нажимаем на новый каталог.
5. “New” → “Python 3”. Открывается новая вкладка браузера с файлом “Untitled” (.ipynb). Закрываем вкладку.
6. В каталоге work1 видим файл Untitled.ipynb ставим возле него галочку → нажимаем на появившуюся вверху слева кнопку “Shutdown” (остановить выполнение).

7. Снова файл Untitled.ipynb ставим возле него галочку → нажимаем на появившуюся вверху слева кнопку “Rename” → вводим имя файла (work1.ipynb).

8. Нажимаем на файл work1.ipynb открывается окно в новой вкладке браузера (рис.).

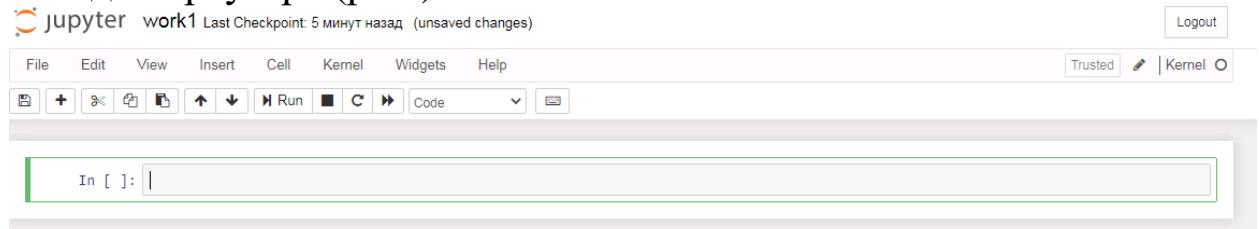


Рисунок – Jupyter Notebook work1.ipynb

8. Текст программ в Jupyter Notebook разделен на блоки. Каждый блок выполняется отдельно – для этого нажимается сочетание клавиш Shift + Enter (рис.). После выполнения блока ему присваивается номер (“In [1]”) и автоматически создается новый блок (также его можно создать кнопкой “+” в верхней панели инструментов). Результат сохраняется в оперативной памяти.



Рисунок – Jupyter Notebook work1.ipynb

9. Можно вернуться к выполненному блоку и выполнить его повторно (номер блока инкрементируется, рис.).

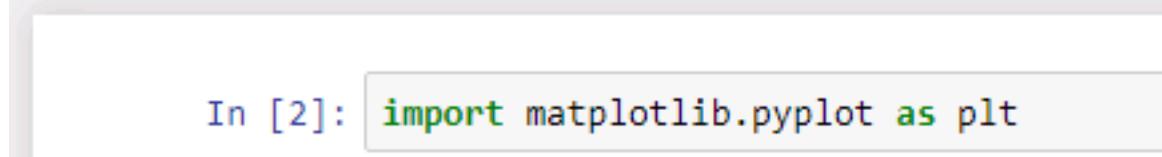


Рисунок – Jupyter Notebook work1.ipynb

10. Очистка оперативной памяти для данной программы производится перезагрузкой ядра (кнопка  на панели инструментов).

11. Разделение на блоки удобно, когда требуется провести анализ обработки одних и тех же данных различными алгоритмами (рис.).

12. Необходимо проверить наличие необходимых библиотек (рис.).

```
In [1]: import numpy as np # импорт библиотеки нужен, чтобы можно было работать с ее функциями
# импортируется каждая библиотека отдельно, чтобы все не занимали слишком много ОЗУ
# библиотека питрп используется для быстрой работы с числами и массивами чисел
# as np - далее в тексте программы будем ссылаться на эту библиотеку через имя np
# np - общепринятый акроним питрп
```

```
In [2]: x = np.array([[1, 2, 3], [4, 5, 6]]) # создаем питрп-массив 2x3
```

```
In [3]: print("x:\n{}".format(x)) # выводим значения массива с отделением его имени от значений знаком \n - перенос на новую строку
```

```
x:
[[1 2 3]
 [4 5 6]]
```

```
In [4]: print("x-array:{}\n".format(x)) # выводим значения массива без отделением его имени от значений
```

```
x-array: [[1 2 3]
 [4 5 6]]
```

```
In [ ]:
```

Рисунок – Jupyter Notebook work1.ipynb

```
In [1]: import numpy as np # work with numbers
import pandas as pd # work with various data
import matplotlib.pyplot as plt # work with graphs
import sklearn as skl # machine learning library|
```

Рисунок – Проверка наличия библиотек (должно выполниться без ошибок)

```
In [2]: arr = np.random.rand(15,2) # создаем массив случайных чисел - 15x2
```

```
In [3]: plt.figure(figsize=(20,5)) # создаем область для рисования размером 20x5
plt.xlabel('point number') # задаем название горизонтальной оси
plt.ylabel('point value') # задаем название вертикальной оси
plt.plot(arr) # выводим графики на экран
```

```
Out[3]: [<matplotlib.lines.Line2D at 0xa146352e8>,
<matplotlib.lines.Line2D at 0xa146354e0>]
```

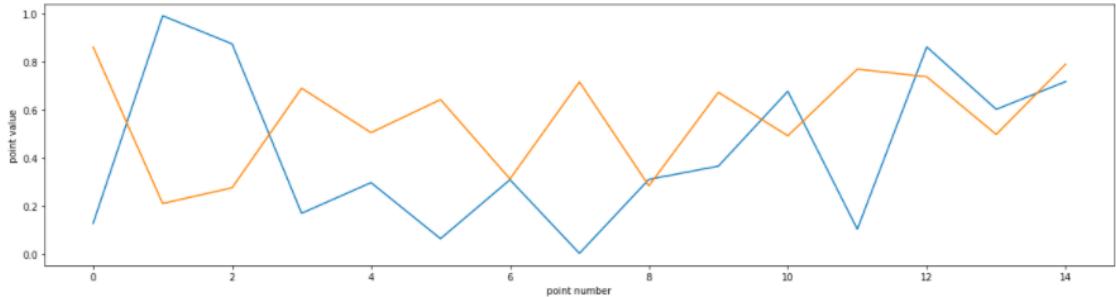


Рисунок – Программа для выполнения и отчета

Часть 1

Logistic Regression with Python

Теоретические сведения

In this notebook, you will learn Logistic Regression, and then, you'll create a model for a telecommunication company, to predict when its customers will leave for a competitor, so that they can take some action to retain the customers.

В этой практической работе вы изучите логистическую регрессию, а затем создадите модель для телекоммуникационной компании, чтобы предсказать, когда ее клиенты уйдут к конкуренту, чтобы они могли предпринять какие-либо действия для удержания клиентов.

Программа работы

1. [About the dataset](#)
2. [Data pre-processing and selection](#)
3. [Modeling \(Logistic Regression with Scikit-learn\)](#)
4. [Evaluation](#)
5. [Practice](#)

What is the difference between Linear and Logistic Regression?

While Linear Regression is suited for estimating continuous values (e.g. estimating house price), it is not the best tool for predicting the class of an observed data point. In order to estimate the class of a data point, we need some sort of guidance on what would be the **most probable class** for that data point. For this, we use **Logistic Regression**.

Recall linear regression:

As you know, **Linear regression** finds a function that relates a continuous dependent variable, y , to some predictors (independent variables x_1, x_2 , etc.). For example, Simple linear regression assumes a function of the form:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

and finds the values of parameters $\theta_0, \theta_1, \theta_2$, etc, where the term θ_0 is the "intercept". It can be generally shown as:

$$h_{\theta}(x) = \theta^T X$$

Logistic Regression is a variation of Linear Regression, useful when the observed dependent variable, \mathbf{y} , is categorical. It produces a formula that predicts the probability of the class label as a function of the independent variables.

Logistic regression fits a special s-shaped curve by taking the linear regression and transforming the numeric estimate into a probability with the following function, which is called sigmoid function σ :

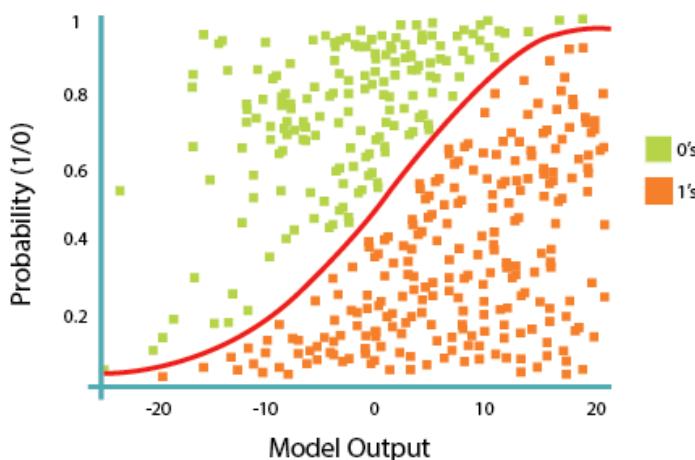
$$h_{\theta}(x) = \sigma(\theta^T X) = \frac{e^{(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots)}}{1 + e^{(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots)}}$$

Or:

$$\text{ProbabilityOfaClass}_1 = P(Y = 1|X) = \sigma(\theta^T X) = \frac{e^{\theta^T X}}{1 + e^{\theta^T X}}$$

In this equation, $\theta^T X$ is the regression result (the sum of the variables weighted by the coefficients), `exp` is the exponential function and $\sigma(\theta^T X)$ is the sigmoid or [logistic function](#), also called logistic curve. It is a common "S" shape (sigmoid curve).

So, briefly, Logistic Regression passes the input through the logistic/sigmoid but then treats the result as a probability:



The objective of **Logistic Regression** algorithm, is to find the best parameters θ , for $h_{\theta}(x) = \sigma(\theta^T X)$, in such a way that the model best predicts the class of each case.

Customer churn with Logistic Regression

A telecommunications company is concerned about the number of customers leaving their land-line business for cable competitors. They need to understand who is leaving. Imagine that you are an

analyst at this company and you have to find out who is leaving and why.

Lets first import required libraries:

```
In [1]: import pandas as pd  
import pylab as pl  
import numpy as np  
import scipy.optimize as opt  
from sklearn import preprocessing  
%matplotlib inline  
import matplotlib.pyplot as plt
```

About the dataset

We will use a telecommunications dataset for predicting customer churn. This is a historical customer dataset where each row represents one customer. The data is relatively easy to understand, and you may uncover insights you can use immediately. Typically it is less expensive to keep customers than acquire new ones, so the focus of this analysis is to predict the customers who will stay with the company.

This data set provides information to help you predict what behavior will help you to retain customers. You can analyze all relevant customer data and develop focused customer retention programs.

The dataset includes information about:

- Customers who left within the last month – the column is called Churn
- Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
- Customer account information – how long they had been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- Demographic info about customers – gender, age range, and if they have partners and dependents

Load the Telco Churn data

Telco Churn is a hypothetical data file that concerns a telecommunications company's efforts to reduce turnover in its customer base. Each case corresponds to a separate customer and it records various demographic and service usage information. Before you can work with the data, you must use the URL to get the ChurnData.csv.

To download the data, we will use `!wget` to download it from IBM Object Storage.

```
In [2]: #Click here and press Shift+Enter  
#!wget -O ChurnData.csv https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-dat
```

Load Data From CSV File

```
In [3]: churn_df = pd.read_csv("ChurnData.csv")  
churn_df.head()
```

```
Out[3]:   tenure  age  address  income  ed  employ  equip  callcard  wireless  longmon ...  pager  internet  c  
0      11.0  33.0       7.0    136.0  5.0      5.0     0.0      1.0      1.0     4.40 ...     1.0      0.0
```

1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	9.45	...	0.0	0.0	
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	6.30	...	0.0	0.0	
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	6.05	...	1.0	1.0	
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	0.0	0.0

5 rows × 28 columns

Data pre-processing and selection

Lets select some features for the modeling. Also we change the target data type to be integer, as it is a requirement by the sklearn algorithm:

```
In [4]: churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip', 'churn']]
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

```
Out[4]:
```

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

Practice

How many rows and columns are in this dataset in total? What are the name of columns?

```
In [5]: # write your code here
```

Double-click **here** for the solution.

Lets define X, and y for our dataset:

```
In [6]: X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']])
X[0:5]
```

```
Out[6]: array([[ 11.,   33.,    7.,  136.,    5.,    5.,    0.],
       [ 33.,   33.,   12.,   33.,    2.,    0.,    0.],
       [ 23.,   30.,    9.,   30.,    1.,    2.,    0.],
       [ 38.,   35.,    5.,   76.,    2.,   10.,    1.],
       [  7.,   35.,   14.,   80.,    2.,   15.,    0.]])
```

```
In [7]: y = np.asarray(churn_df['churn'])
y[0:5]
```

```
Out[7]: array([1, 1, 0, 0, 0])
```

Also, we normalize the dataset:

```
In [8]: from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
```

```
x[0:5]
```

```
Out[8]: array([[-1.13518441, -0.62595491, -0.4588971 ,  0.4751423 ,  1.6961288 ,
   -0.58477841, -0.85972695],
   [-0.11604313, -0.62595491,  0.03454064, -0.32886061, -0.6433592 ,
   -1.14437497, -0.85972695],
   [-0.57928917, -0.85594447, -0.261522 , -0.35227817, -1.42318853,
   -0.92053635, -0.85972695],
   [ 0.11557989, -0.47262854, -0.65627219,  0.00679109, -0.6433592 ,
   -0.02518185,  1.16316 ],
   [-1.32048283, -0.47262854,  0.23191574,  0.03801451, -0.6433592 ,
   0.53441472, -0.85972695]])
```

Train/Test dataset

Okay, we split our dataset into train and test set:

```
In [9]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( x, y, test_size=0.2, random_state=4
print ('Train set:', X_train.shape, y_train.shape)
print ('Test set:', X_test.shape, y_test.shape)

Train set: (160, 7) (160,)
Test set: (40, 7) (40,)
```

Modeling (Logistic Regression with Scikit-learn)

Lets build our model using **LogisticRegression** from Scikit-learn package. This function implements logistic regression and can use different numerical optimizers to find parameters, including 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' solvers. You can find extensive information about the pros and cons of these optimizers if you search it in internet.

The version of Logistic Regression in Scikit-learn, support regularization. Regularization is a technique used to solve the overfitting problem in machine learning models. **C** parameter indicates **inverse of regularization strength** which must be a positive float. Smaller values specify stronger regularization.

Now lets fit our model with train set:

```
In [10]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
LR

Out[10]: LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                           intercept_scaling=1, l1_ratio=None, max_iter=100,
                           multi_class='warn', n_jobs=None, penalty='l2',
                           random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                           warm_start=False)
```

Now we can predict using our test set:

```
In [11]: yhat = LR.predict(X_test)
yhat

Out[11]: array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0])
```

predict_proba returns estimates for all classes, ordered by the label of classes. So, the first column is the probability of class 1, $P(Y=1|X)$, and second column is probability of class 0, $P(Y=0|X)$:

```
In [12]: yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

```
Out[12]: array([[0.54132919, 0.45867081],
 [0.60593357, 0.39406643],
 [0.56277713, 0.43722287],
 [0.63432489, 0.36567511],
 [0.56431839, 0.43568161],
 [0.55386646, 0.44613354],
 [0.52237207, 0.47762793],
 [0.60514349, 0.39485651],
 [0.41069572, 0.58930428],
 [0.6333873 , 0.3666127 ],
 [0.58068791, 0.41931209],
 [0.62768628, 0.37231372],
 [0.47559883, 0.52440117],
 [0.4267593 , 0.5732407 ],
 [0.66172417, 0.33827583],
 [0.55092315, 0.44907685],
 [0.51749946, 0.48250054],
 [0.485743 , 0.514257 ],
 [0.49011451, 0.50988549],
 [0.52423349, 0.47576651],
 [0.61619519, 0.38380481],
 [0.52696302, 0.47303698],
 [0.63957168, 0.36042832],
 [0.52205164, 0.47794836],
 [0.50572852, 0.49427148],
 [0.70706202, 0.29293798],
 [0.55266286, 0.44733714],
 [0.52271594, 0.47728406],
 [0.51638863, 0.48361137],
 [0.71331391, 0.28668609],
 [0.67862111, 0.32137889],
 [0.50896403, 0.49103597],
 [0.42348082, 0.57651918],
 [0.71495838, 0.28504162],
 [0.59711064, 0.40288936],
 [0.63808839, 0.36191161],
 [0.39957895, 0.60042105],
 [0.52127638, 0.47872362],
 [0.65975464, 0.34024536],
 [0.5114172 , 0.4885828 ]])
```

Evaluation

jaccard index

Lets try jaccard index for accuracy evaluation. we can define jaccard as the size of the intersection divided by the size of the union of two label sets. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

```
In [13]: from sklearn.metrics import jaccard_similarity_score
jaccard_similarity_score(y_test, yhat)
```

```
J:\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:635: DeprecationWarning:
jaccard_similarity_score has been deprecated and replaced with jaccard_score. It will
be removed in version 0.23. This implementation has surprising behavior for binary and m
ulticlass classification tasks.
```

```
'and multiclass classification tasks.', DeprecationWarning)
```

```
Out[13]: 0.75
```

confusion matrix

Another way of looking at accuracy of classifier is to look at **confusion matrix**.

In [14]:

```
from sklearn.metrics import classification_report, confusion_matrix
import itertools
def plot_confusion_matrix(cm, classes,
                         normalize=False,
                         title='Confusion matrix',
                         cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

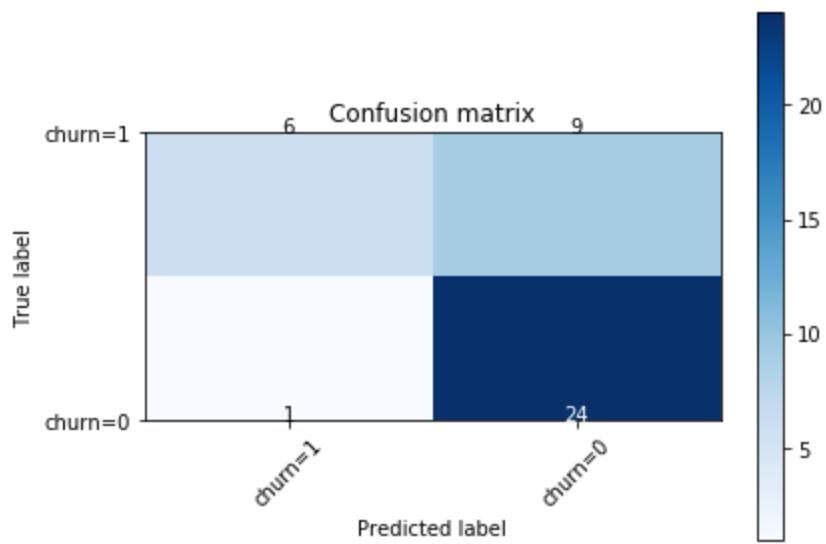
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
[[ 6  9]
 [ 1 24]]
```

In [15]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1', 'churn=0'], normalize=False, title='Confusion matrix, without normalization')
[[ 6  9]
 [ 1 24]]
```



Look at first row. The first row is for customers whose actual churn value in test set is 1. As you can calculate, out of 40 customers, the churn value of 15 of them is 1. And out of these 15, the classifier correctly predicted 6 of them as 1, and 9 of them as 0.

It means, for 6 customers, the actual churn value were 1 in test set, and classifier also correctly predicted those as 1. However, while the actual label of 9 customers were 1, the classifier predicted those as 0, which is not very good. We can consider it as error of the model for first row.

What about the customers with churn value 0? Lets look at the second row. It looks like there were 25 customers whom their churn value were 0.

The classifier correctly predicted 24 of them as 0, and one of them wrongly as 1. So, it has done a good job in predicting the customers with churn value 0. A good thing about confusion matrix is that shows the model's ability to correctly predict or separate the classes. In specific case of binary classifier, such as this example, we can interpret these numbers as the count of true positives, false positives, true negatives, and false negatives.

```
In [16]: print (classification_report(y_test, yhat))
```

	precision	recall	f1-score	support
0	0.73	0.96	0.83	25
1	0.86	0.40	0.55	15
accuracy			0.75	40
macro avg	0.79	0.68	0.69	40
weighted avg	0.78	0.75	0.72	40

Based on the count of each section, we can calculate precision and recall of each label:

- **Precision** is a measure of the accuracy provided that a class label has been predicted. It is defined by: $\text{precision} = \text{TP} / (\text{TP} + \text{FP})$
- **Recall** is true positive rate. It is defined as: $\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$

So, we can calculate precision and recall of each class.

F1 score: Now we are in the position to calculate the F1 scores for each label based on the precision and recall of that label.

The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classifier has a good value for both recall and precision.

And finally, we can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.72 in our case.

log loss

Now, lets try **log loss** for evaluation. In logistic regression, the output can be the probability of customer churn is yes (or equals to 1). This probability is a value between 0 and 1. Log loss(Logarithmic loss) measures the performance of a classifier where the predicted output is a probability value between 0 and 1.

```
In [17]: from sklearn.metrics import log_loss  
log_loss(y_test, yhat_prob)
```

```
Out[17]: 0.6017092478101187
```

Practice

Try to build Logistic Regression model again for the same dataset, but this time, use different **solver** and **regularization** values? What is new **logLoss** value?

```
In [18]: # write your code here
```

Double-click **here** for the solution.

Часть 2

Collaborative filtering

Теоретические сведения

Recommendation systems are a collection of algorithms used to recommend items to users based on information taken from the user. These systems have become ubiquitous can be commonly seen in online stores, movies databases and job finders. In this notebook, we will explore recommendation systems based on Collaborative Filtering and implement simple version of one using Python and the Pandas library.

Рекомендательные системы представляют собой набор алгоритмов, используемых для рекомендации чего-либо пользователям на основе информации, полученной от пользователя. Эти системы стали повсеместными, их можно часто увидеть в интернет-магазинах, базах данных фильмов и программах поиска работы. В этой лабораторной работе мы рассмотрим системы рекомендаций, основанные на совместной фильтрации, и реализуем простую версию такой системы с использованием Python и библиотеки Pandas.

Программа работы

1. [Acquiring the Data](#)
2. [Preprocessing](#)
3. [Collaborative Filtering](#)

Acquiring the Data

To acquire and extract the data, simply run the following Bash scripts:

```
In [1]: #!wget -O moviedataset.zip https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-#print('unzipping ...')#!unzip -o -j moviedataset.zip
```

Now you're ready to start working with the data!

Preprocessing

First, let's get all of the imports out of the way:

```
In [2]: #Dataframe manipulation library
import pandas as pd
#Math functions, we'll only need the sqrt function so let's import only that
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Now let's read each file into their Dataframes:

```
In [3]: #Storing the movie information into a pandas dataframe
movies_df = pd.read_csv('movies.csv')
#Storing the user information into a pandas dataframe
ratings_df = pd.read_csv('ratings.csv')
```

Let's also take a peek at how each of them are organized:

```
In [4]: #Head is a function that gets the first N rows of a dataframe. N's default is 5.
movies_df.head()
```

	movielid	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

So each movie has a unique ID, a title with its release year along with it (Which may contain unicode characters) and several different genres in the same field. Let's remove the year from the title column and place it into its own one by using the handy `extract` function that Pandas has.

Let's remove the year from the `title` column by using pandas' `replace` function and store in a new `year` column.

```
In [5]: #Using regular expressions to find a year stored between parentheses
#We specify the parentheses so we don't conflict with movies that have years in their ti
movies_df['year'] = movies_df.title.str.extract('(\d\d\d\d)', expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)', expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\d\d\d\d)', '')
#Applying the strip function to get rid of any ending whitespace characters that may hav
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
```

Let's look at the result!

```
In [6]: movies_df.head()
```

	movielid	title	genres	year
0	1	Toy Story	Adventure Animation Children Comedy Fantasy	1995

1	2	Jumanji	Adventure Children Fantasy	1995
2	3	Grumpier Old Men	Comedy Romance	1995
3	4	Waiting to Exhale	Comedy Drama Romance	1995
4	5	Father of the Bride Part II	Comedy	1995

With that, let's also drop the genres column since we won't need it for this particular recommendation system.

```
In [7]: #Dropping the genres column
movies_df = movies_df.drop('genres', 1)
```

Here's the final movies dataframe:

```
In [8]: movies_df.head()
```

	moviedb_id	title	year
0	1	Toy Story	1995
1	2	Jumanji	1995
2	3	Grumpier Old Men	1995
3	4	Waiting to Exhale	1995
4	5	Father of the Bride Part II	1995

Next, let's look at the ratings dataframe.

```
In [9]: ratings_df.head()
```

	userId	moviedb_id	rating	timestamp
0	1	169	2.5	1204927694
1	1	2471	3.0	1204927438
2	1	48516	5.0	1204927435
3	2	2571	3.5	1436165433
4	2	109487	4.0	1436165496

Every row in the ratings dataframe has a user id associated with at least one movie, a rating and a timestamp showing when they reviewed it. We won't be needing the timestamp column, so let's drop it to save on memory.

```
In [10]: #Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
```

Here's how the final ratings Dataframe looks like:

```
In [11]: ratings_df.head()
```

	userId	moviedb_id	rating
--	--------	------------	--------

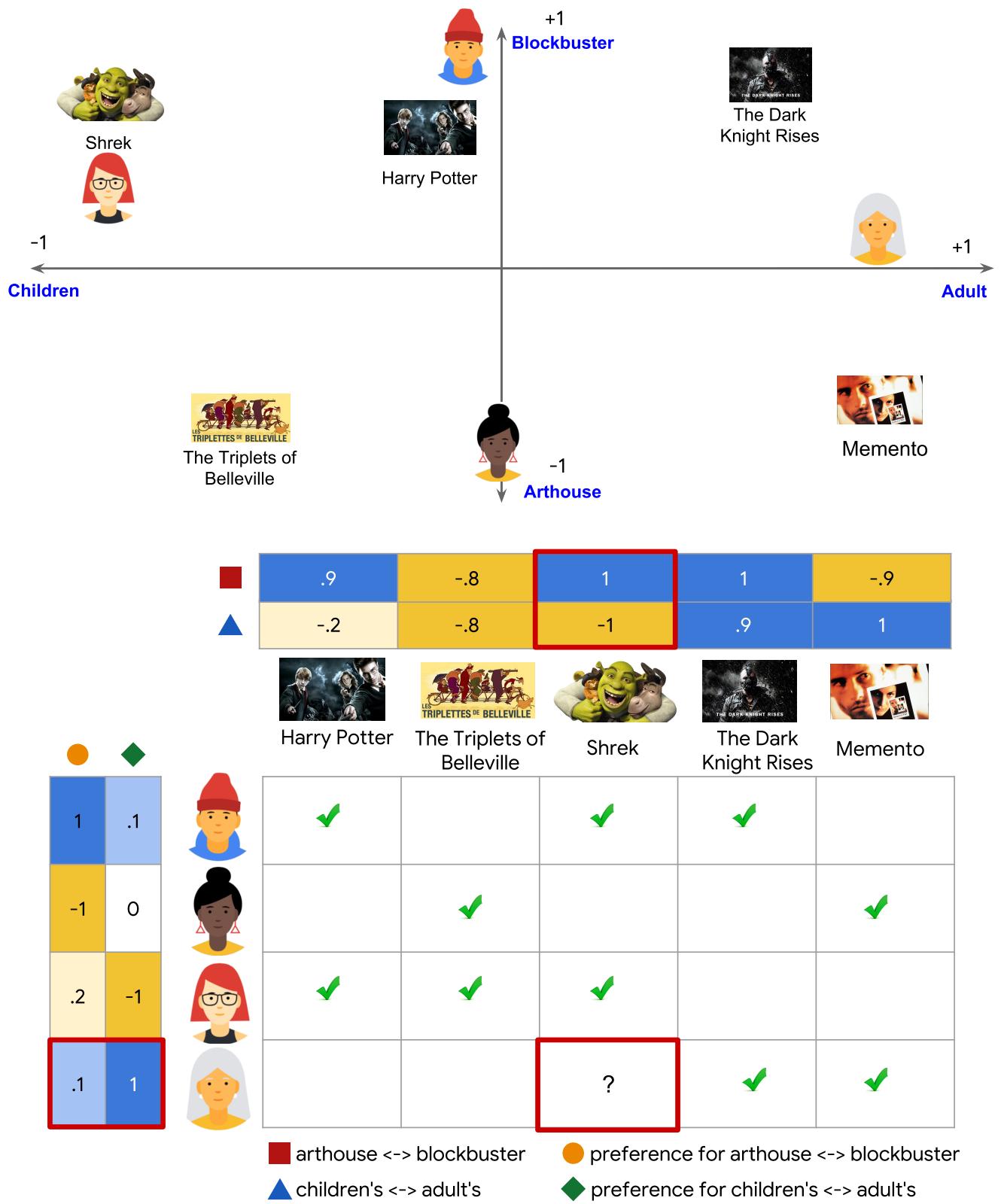
0	1	169	2.5
1	1	2471	3.0
2	1	48516	5.0
3	2	2571	3.5
4	2	109487	4.0

Collaborative Filtering

Now, time to start our work on recommendation systems.

The first technique we're going to take a look at is called **Collaborative Filtering**, which is also known as **User-User Filtering**. As hinted by its alternate name, this technique uses other users to recommend items to the input user. It attempts to find users that have similar preferences and opinions as the input and then recommends items that they have liked to the input.





There are several methods of finding similar users (Even some making use of Machine Learning), and the one we will be using here is going to be based on the **Pearson Correlation Function**.

The process for creating a User Based recommendation system is as follows:

- Select a user with the movies the user has watched
- Based on his rating to movies, find the top X neighbours
- Get the watched movie record of the user for each neighbour.
- Calculate a similarity score using some formula
- Recommend the items with the highest score

Let's begin by creating an input user to recommend movies to:

Notice: To add more movies, simply increase the amount of elements in the userInput. Feel free to add more in! Just be sure to write it in with capital letters and if a movie starts with a "The", like "The Matrix" then write it in like this: 'Matrix, The' .

Первый метод, который мы собираемся рассмотреть, называется колаборативной фильтрацией, также известной как фильтрация пользователей. Как следует из его альтернативного имени, этот метод использует других пользователей, чтобы рекомендовать элементы пользователю ввода. Он пытается найти пользователей со схожими предпочтениями и мнениями, а затем рекомендует элементы, которые им понравились. Существует несколько методов поиска похожих пользователей (даже некоторые с использованием машинного обучения), и тот, который мы будем использовать здесь, будет основан на корреляционной функции Пирсона.

Процесс создания системы рекомендаций на основе пользователей выглядит следующим образом:

- Выберите пользователя с фильмами, которые он смотрел
- Основываясь на его рейтинге фильмов, найдите X лучших соседей.
- Получите запись просмотренного фильма пользователя для каждого соседа.
- Вычислите показатель сходства, используя некоторую формулу
- Рекомендуйте предметы с наивысшим баллом

Начнем с создания входного пользователя, которому мы будем рекомендовать фильмы:

Примечание. Чтобы добавить больше фильмов, просто увеличьте количество элементов в файле userInput. Это рекомендуется попробовать Не забудьте написать это заглавными буквами, и если фильм начинается с «The», например «Матрица», напишите его так: «Матрица, The».

```
In [12]: userInput = [
    {'title':'Breakfast Club, The', 'rating':5},
    {'title':'Toy Story', 'rating':3.5},
    {'title':'Jumanji', 'rating':2},
    {'title':"Pulp Fiction", 'rating':5},
    {'title':'Akira', 'rating':4.5}
]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

```
Out[12]:
```

	title	rating
0	Breakfast Club, The	5.0
1	Toy Story	3.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	4.5

Add movield to input user

With the input complete, let's extract the input movies's ID's from the movies dataframe and add them into it.

We can achieve this by first filtering out the rows that contain the input movies' title and then merging this subset with the input dataframe. We also drop unnecessary columns for the input to save memory space.

In [13]:

```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

Out[13]:

	moviedb	title	rating
0	1	Toy Story	3.5
1	2	Jumanji	2.0
2	296	Pulp Fiction	5.0
3	1274	Akira	4.5
4	1968	Breakfast Club, The	5.0

The users who has seen the same movies

Now with the movie ID's in our input, we can now get the subset of users that have watched and reviewed the movies in our input.

In [14]:

```
#Filtering out users that have watched movies that the input has watched and storing it
userSubset = ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]
userSubset.head()
```

Out[14]:

	userId	moviedb	rating
19	4	296	4.0
441	12	1968	3.0
479	13	2	2.0
531	13	1274	5.0
681	14	296	2.0

We now group up the rows by user ID.

In [15]:

```
#Groupby creates several sub dataframes where they all have the same value in the column
userSubsetGroup = userSubset.groupby(['userId'])
```

lets look at one of the users, e.g. the one with userID=1130

In [16]:

```
userSubsetGroup.get_group(1130)
```

Out[16]:

	userId	moviedb	rating
104167	1130	1	0.5
104168	1130	2	4.0

104214	1130	296	4.0
104363	1130	1274	4.5
104443	1130	1968	4.5

Let's also sort these groups so the users that share the most movies in common with the input have higher priority. This provides a richer recommendation since we won't go through every single user.

```
In [17]: #Sorting it so users with movie most in common with the input will have priority
userSubsetGroup = sorted(userSubsetGroup, key=lambda x: len(x[1]), reverse=True)
```

Now lets look at the first user

```
In [18]: userSubsetGroup[0:3]
```

```
Out[18]: [(75,          userId  movieId  rating
 7507        75        1    5.0
 7508        75        2    3.5
 7540        75      296    5.0
 7633        75      1274    4.5
 7673        75      1968  5.0), (106,          userId  movieId  rating
 9083       106        1    2.5
 9084       106        2    3.0
 9115       106      296    3.5
 9198       106      1274    3.0
 9238       106      1968  3.5), (686,          userId  movieId  rating
 61336      686        1    4.0
 61337      686        2    3.0
 61377      686      296    4.0
 61478      686      1274    4.0
 61569      686      1968  5.0)]
```

Similarity of users to input user

Next, we are going to compare all users (not really all !!!) to our specified user and find the one that is most similar.

we're going to find out how similar each user is to the input through the **Pearson Correlation**

Coefficient. It is used to measure the strength of a linear association between two variables. The formula for finding this coefficient between sets X and Y with N values can be seen in the image below.

Why Pearson Correlation?

Pearson correlation is invariant to scaling, i.e. multiplying all elements by a nonzero constant or adding any constant to all elements. For example, if you have two vectors X and Y, then, `pearson(X, Y) == pearson(X, 2 * Y + 3)`. This is a pretty important property in recommendation systems because for example two users might rate two series of items totally different in terms of absolute rates, but they would be similar users (i.e. with similar ideas) with similar rates in various scales .

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

The values given by the formula vary from $r = -1$ to $r = 1$, where 1 forms a direct correlation between the two entities (it means a perfect positive correlation) and -1 forms a perfect negative correlation.

In our case, a 1 means that the two users have similar tastes while a -1 means the opposite.

We will select a subset of users to iterate through. This limit is imposed because we don't want to waste too much time going through every single user.

In [19]: `userSubsetGroup = userSubsetGroup[0:100]`

Now, we calculate the Pearson Correlation between input user and subset group, and store it in a dictionary, where the key is the user Id and the value is the coefficient

In [20]: `#Store the Pearson Correlation in a dictionary, where the key is the user Id and the val
pearsonCorrelationDict = {}

#For every user group in our subset
for name, group in userSubsetGroup:
 #Let's start by sorting the input and current user group so the values aren't mixed
 group = group.sort_values(by='movieId')
 inputMovies = inputMovies.sort_values(by='movieId')
 #Get the N for the formula
 nRatings = len(group)
 #Get the review scores for the movies that they both have in common
 temp_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]
 #And then store them in a temporary buffer variable in a list format to facilitate f
 tempRatingList = temp_df['rating'].tolist()
 #Let's also put the current user group reviews in a list format
 tempGroupList = group['rating'].tolist()
 #Now let's calculate the pearson correlation between two users, so called, x and y
 Sxx = sum([i**2 for i in tempRatingList]) - pow(sum(tempRatingList), 2)/float(nRatings)
 Syy = sum([i**2 for i in tempGroupList]) - pow(sum(tempGroupList), 2)/float(nRatings)
 Sxy = sum(i*j for i, j in zip(tempRatingList, tempGroupList)) - sum(tempRatingList)

 #If the denominator is different than zero, then divide, else, 0 correlation.
 if Sxx != 0 and Syy != 0:
 pearsonCorrelationDict[name] = Sxy/sqrt(Sxx*Syy)
 else:
 pearsonCorrelationDict[name] = 0`

In [21]: `pearsonCorrelationDict.items()`

Out[21]: `dict_items([(75, 0.8272781516947562), (106, 0.5860090386731182), (686, 0.832050294337843
7), (815, 0.5765566601970551), (1040, 0.9434563530497265), (1130, 0.2891574659831201),
(1502, 0.8770580193070299), (1599, 0.4385290096535153), (1625, 0.716114874039432), (195
0, 0.179028718509858), (2065, 0.4385290096535153), (2128, 0.5860090386731196), (2432, 0.
1386750490563073), (2791, 0.8770580193070299), (2839, 0.8204126541423674), (2948, -0.117
20180773462392), (3025, 0.45124262819713973), (3040, 0.89514359254929), (3186, 0.6784622
064861935), (3271, 0.26989594817970664), (3429, 0.0), (3734, -0.15041420939904673), (409
9, 0.05860090386731196), (4208, 0.29417420270727607), (4282, -0.4385290096535115), (429
2, 0.6564386345361464), (4415, -0.11183835382312353), (4586, -0.9024852563942795), (472
5, -0.08006407690254357), (4818, 0.4885967564883424), (5104, 0.7674257668936507), (5165,
-0.4385290096535153), (5547, 0.17200522903844556), (6082, -0.04728779924109591), (6207,
0.9615384615384616), (6366, 0.6577935144802716), (6482, 0.0), (6530, -0.351605423203870
9), (7235, 0.6981407669689391), (7403, 0.11720180773462363), (7641, 0.7161148740394331),
(7996, 0.626600514784504), (8008, -0.22562131409856986), (8086, 0.6933752452815365), (82
45, 0.0), (8572, 0.8600261451922278), (8675, 0.5370861555295773), (9101, -0.086002614519
22278), (9358, 0.692178738358485), (9663, 0.193972725041952), (9994, 0.503027272865958
7), (10248, -0.24806946917841693), (10315, 0.537086155529574), (10368, 0.468807230938494
5), (10607, 0.41602514716892186), (10707, 0.9615384615384616), (10863, 0.602018301634559
5), (11314, 0.8204126541423654), (11399, 0.517260600111872), (11769, 0.937614461876991
4), (11827, 0.4902903378454601), (12069, 0.0), (12120, 0.9292940047327363), (12211, 0.86
00261451922278), (12325, 0.9616783115081544), (12916, 0.5860090386731196), (12921, 0.661
1073566849309), (13053, 0.9607689228305227), (13142, 0.6016568375961863), (13260, 0.7844
645405527362), (13366, 0.8951435925492911), (13768, 0.8770580193070289), (13888, 0.25087
26030021272), (13923, 0.3516054232038718), (13934, 0.17200522903844556), (14529, 0.74179
01772340937), (14551, 0.537086155529574), (14588, 0.21926450482675766), (14984, 0.716114
874039432), (15137, 0.5860090386731196), (15157, 0.9035841064985974), (15466, 0.72057669`

```
21228921), (15670, 0.516015687115336), (15834, 0.22562131409856986), (16292, 0.657793514  
4802716), (16456, 0.7161148740394331), (16506, 0.5481612620668942), (17246, 0.4803844614  
1526137), (17438, 0.7093169886164387), (17501, 0.8168748513121271), (17502, 0.8272781516  
947562), (17666, 0.7689238340176859), (17735, 0.7042381820123422), (17742, 0.39223227027  
63681), (17757, 0.64657575013984), (17854, 0.537086155529574), (17897, 0.877058019307028  
9), (17944, 0.2713848825944774), (18301, 0.29838119751643016), (18509, 0.132221471336986  
2)])
```

```
In [22]: pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')  
pearsonDF.columns = ['similarityIndex']  
pearsonDF['userId'] = pearsonDF.index  
pearsonDF.index = range(len(pearsonDF))  
pearsonDF.head()
```

Out[22]:

	similarityIndex	userId
0	0.827278	75
1	0.586009	106
2	0.832050	686
3	0.576557	815
4	0.943456	1040

The top x similar users to input user

Now let's get the top 50 users that are most similar to the input.

```
In [23]: topUsers=pearsonDF.sort_values(by='similarityIndex', ascending=False) [0:50]  
topUsers.head()
```

Out[23]:

	similarityIndex	userId
64	0.961678	12325
34	0.961538	6207
55	0.961538	10707
67	0.960769	13053
4	0.943456	1040

Now, let's start recommending movies to the input user.

Rating of selected users to all movies

We're going to do this by taking the weighted average of the ratings of the movies using the Pearson Correlation as the weight. But to do this, we first need to get the movies watched by the users in our **pearsonDF** from the ratings dataframe and then store their correlation in a new column called **_similarityIndex**. This is achieved below by merging of these two tables.

```
In [24]: topUsersRating=topUsers.merge(ratings_df, left_on='userId', right_on='userId', how='inner')  
topUsersRating.head()
```

Out[24]:

	similarityIndex	userId	movieId	rating
0	0.961678	12325	1	3.5
1	0.961678	12325	2	1.5
2	0.961678	12325	3	3.0

3	0.961678	12325	5	0.5
4	0.961678	12325	6	2.5

Now all we need to do is simply multiply the movie rating by its weight (The similarity index), then sum up the new ratings and divide it by the sum of the weights.

We can easily do this by simply multiplying two columns, then grouping up the dataframe by movield and then dividing two columns:

It shows the idea of all similar users to candidate movies for the input user:

```
In [25]: #Multiplies the similarity by the user's ratings
topUsersRating['weightedRating'] = topUsersRating['similarityIndex']*topUsersRating['rating']
topUsersRating.head()
```

	similarityIndex	userId	movield	rating	weightedRating
0	0.961678	12325	1	3.5	3.365874
1	0.961678	12325	2	1.5	1.442517
2	0.961678	12325	3	3.0	2.885035
3	0.961678	12325	5	0.5	0.480839
4	0.961678	12325	6	2.5	2.404196

```
In [26]: #Applies a sum to the topUsers after grouping it up by userId
tempTopUsersRating = topUsersRating.groupby('movield').sum()[['similarityIndex','weightedRating']]
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating.head()
```

movield	sum_similarityIndex	sum_weightedRating
1	38.376281	140.800834
2	38.376281	96.656745
3	10.253981	27.254477
4	0.929294	2.787882
5	11.723262	27.151751

```
In [27]: #Creates an empty dataframe
recommendation_df = pd.DataFrame()
#Now we take the weighted average
recommendation_df['weighted average recommendation score'] = tempTopUsersRating['sum_weightedRating']/tempTopUsersRating['sum_similarityIndex']
recommendation_df['movield'] = tempTopUsersRating.index
recommendation_df.head()
```

movield	weighted average recommendation score	movield
1	3.668955	1
2	2.518658	2
3	2.657941	3
4	3.000000	4

Now let's sort it and see the top 20 movies that the algorithm recommended!

```
In [28]: recommendation_df = recommendation_df.sort_values(by='weighted average recommendation score')
recommendation_df.head(10)
```

```
Out[28]: weighted average recommendation score  movield
```

movield			
5073		5.0	5073
3329		5.0	3329
2284		5.0	2284
26801		5.0	26801
6776		5.0	6776
6672		5.0	6672
3759		5.0	3759
3769		5.0	3769
3775		5.0	3775
90531		5.0	90531

```
In [29]: movies_df.loc[movies_df['movieId'].isin(recommendation_df.head(10) ['movieId'].tolist())]
```

```
Out[29]: movield  title  year
```

	movield	title	year
2200	2284	Bandit Queen	1994
3243	3329	Year My Voice Broke, The	1987
3669	3759	Fun and Fancy Free	1947
3679	3769	Thunderbolt and Lightfoot	1974
3685	3775	Make Mine Music	1946
4978	5073	Son's Room, The (Stanza del figlio, La)	2001
6563	6672	War Photographer	2001
6667	6776	Lagaan: Once Upon a Time in India	2001
9064	26801	Dragon Inn (Sun lung moon hak chan)	1992
18106	90531	Shame	2011

Advantages and Disadvantages of Collaborative Filtering

Advantages

- Takes other user's ratings into consideration
- Doesn't need to study or extract information from the recommended item
- Adapts to the user's interests which might change over time

Disadvantages

- Approximation function can be slow

- There might be a low of amount of users to approximate
- Privacy issues when trying to learn the user's preferences

Часть 3

Content-based filtering

Программа работы

1. Acquiring the Data
2. Preprocessing
3. Content-Based Filtering

Acquiring the Data

To acquire and extract the data, simply run the following Bash scripts:

```
In [1]: #!wget -O moviedataset.zip https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-unzip('unzipping ...')#!unzip -o -j moviedataset.zip
```

Now you're ready to start working with the data!

Preprocessing

First, let's get all of the imports out of the way:

```
In [2]: #Dataframe manipulation libraryimport pandas as pd#Math functions, we'll only need the sqrt function so let's import only thatfrom math import sqrtimport numpy as npimport matplotlib.pyplot as plt%matplotlib inline
```

Now let's read each file into their Dataframes:

```
In [3]: #Storing the movie information into a pandas dataframemovies_df = pd.read_csv('movies.csv')#Storing the user information into a pandas dataframeratings_df = pd.read_csv('ratings.csv')#Head is a function that gets the first N rows of a dataframe. N's default is 5.movies_df.head()
```

Out[3]:	movield	title	genres
---------	---------	-------	--------

0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

Let's also remove the year from the **title** column by using pandas' replace function and store in a new **year** column.

In [4]:

```
#Using regular expressions to find a year stored between parentheses
#We specify the parentheses so we don't conflict with movies that have years in their title
movies_df['year'] = movies_df.title.str.extract('(\d\d\d\d)', expand=False)
#Removing the parentheses
movies_df['year'] = movies_df.year.str.extract('(\d\d\d\d)', expand=False)
#Removing the years from the 'title' column
movies_df['title'] = movies_df.title.str.replace('(\d\d\d\d)', '')
#Applying the strip function to get rid of any ending whitespace characters that may have been added
movies_df['title'] = movies_df['title'].apply(lambda x: x.strip())
movies_df.head()
```

Out [4]:

	moviedb_id	title	genres	year
0	1	Toy Story	Adventure Animation Children Comedy Fantasy	1995
1	2	Jumanji	Adventure Children Fantasy	1995
2	3	Grumpier Old Men	Comedy Romance	1995
3	4	Waiting to Exhale	Comedy Drama Romance	1995
4	5	Father of the Bride Part II	Comedy	1995

With that, let's also split the values in the **Genres** column into a **list of Genres** to simplify future use. This can be achieved by applying Python's split string function on the correct column.

In [5]:

```
#Every genre is separated by a | so we simply have to call the split function on |
movies_df['genres'] = movies_df.genres.str.split('|')
movies_df.head()
```

Out [5]:

	moviedb_id	title	genres	year
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995
1	2	Jumanji	[Adventure, Children, Fantasy]	1995
2	3	Grumpier Old Men	[Comedy, Romance]	1995
3	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995
4	5	Father of the Bride Part II	[Comedy]	1995

Since keeping genres in a list format isn't optimal for the content-based recommendation system technique, we will use the One Hot Encoding technique to convert the list of genres to a vector where each column corresponds to one possible value of the feature. This encoding is needed for feeding categorical data. In this case, we store every different genre in columns that contain either 1 or 0. 1 shows that a movie has that genre and 0 shows that it doesn't. Let's also store this dataframe in another variable since genres won't be important for our first recommendation system.

```
In [6]: #Copying the movie dataframe into a new one since we won't need to use the genre information
moviesWithGenres_df = movies_df.copy()

#For every row in the dataframe, iterate through the list of genres and place a 1 into the corresponding column
for index, row in movies_df.iterrows():
    for genre in row['genres']:
        moviesWithGenres_df.at[index, genre] = 1
#Filling in the NaN values with 0 to show that a movie doesn't have that column's genre
moviesWithGenres_df = moviesWithGenres_df.fillna(0)
moviesWithGenres_df.head()
```

Out[6]:

		movielid	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance	...
0	1	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0	0.0	...
1	2	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0	0.0	...
2	3	3	Grumpier Old Men	[Comedy, Romance]	1995	0.0	0.0	0.0	1.0	0.0	1.0	...
3	4	4	Waiting to Exhale	[Comedy, Drama, Romance]	1995	0.0	0.0	0.0	1.0	0.0	1.0	...
4	5	5	Father of the Bride Part II	[Comedy]	1995	0.0	0.0	0.0	1.0	0.0	0.0	...

5 rows x 24 columns

Next, let's look at the ratings dataframe.

In [7]: ratings_df.head()

Out[7]:

	userId	movielid	rating	timestamp
0	1	169	2.5	1204927694
1	1	2471	3.0	1204927438
2	1	48516	5.0	1204927435
3	2	2571	3.5	1436165433
4	2	109487	4.0	1436165496

Every row in the ratings dataframe has a user id associated with at least one movie, a rating and a timestamp showing when they reviewed it. We won't be needing the timestamp column, so let's drop it to save on memory.

In [8]:

```
#Drop removes a specified row or column from a dataframe
ratings_df = ratings_df.drop('timestamp', 1)
ratings_df.head()
```

Out[8]:

	userId	movielid	rating
0	1	169	2.5

1	1	2471	3.0
2	1	48516	5.0
3	2	2571	3.5
4	2	109487	4.0

Content-Based recommendation system

Now, let's take a look at how to implement **Content-Based** or **Item-Item recommendation systems**. This technique attempts to figure out what a user's favourite aspects of an item is, and then recommends items that present those aspects. In our case, we're going to try to figure out the input's favorite genres from the movies and ratings given.

Let's begin by creating an input user to recommend movies to:

Notice: To add more movies, simply increase the amount of elements in the **userInput**. Feel free to add more in! Just be sure to write it in with capital letters and if a movie starts with a "The", like "The Matrix" then write it in like this: 'Matrix, The' .

Рассмотрим, как внедрить системы рекомендаций **Content-Based** или **Item-Item**. Этот метод пытается выяснить, какие аспекты элемента нравятся пользователю, а затем рекомендует элементы, которые представляют эти аспекты. В нашем случае мы попытаемся выяснить любимые жанры входных данных по фильмам и данным рейтингам.

Примечание. Чтобы добавить больше фильмов, просто увеличьте количество элементов в файле **userInput**. Это рекомендуется попробовать Не забудьте написать это заглавными буквами, и если фильм начинается с «The», например «Матрица», напишите его так: «Матрица, The».

```
In [9]: userInput = [
    {'title':'Breakfast Club, The', 'rating':5},
    {'title':'Toy Story', 'rating':3.5},
    {'title':'Jumanji', 'rating':2},
    {'title':"Pulp Fiction", 'rating':5},
    {'title':'Akira', 'rating':4.5}
]
inputMovies = pd.DataFrame(userInput)
inputMovies
```

	title	rating
0	Breakfast Club, The	5.0
1	Toy Story	3.5
2	Jumanji	2.0
3	Pulp Fiction	5.0
4	Akira	4.5

Add movielid to input user

With the input complete, let's extract the input movie's ID's from the movies dataframe and add them into it.

We can achieve this by first filtering out the rows that contain the input movie's title and then merging this subset with the input dataframe. We also drop unnecessary columns for the input to save memory space.

In [10]:

```
#Filtering out the movies by title
inputId = movies_df[movies_df['title'].isin(inputMovies['title'].tolist())]
#Then merging it so we can get the movieId. It's implicitly merging it by title.
inputMovies = pd.merge(inputId, inputMovies)
#Dropping information we won't use from the input dataframe
inputMovies = inputMovies.drop('genres', 1).drop('year', 1)
#Final input dataframe
#If a movie you added in above isn't here, then it might not be in the original
#dataframe or it might spelled differently, please check capitalisation.
inputMovies
```

Out[10]:

	moviedb	title	rating
0	1	Toy Story	3.5
1	2	Jumanji	2.0
2	296	Pulp Fiction	5.0
3	1274	Akira	4.5
4	1968	Breakfast Club, The	5.0

We're going to start by learning the input's preferences, so let's get the subset of movies that the input has watched from the Dataframe containing genres defined with binary values.

In [11]:

```
#Filtering out the movies from the input
userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['movieId'])]
userMovies
```

Out[11]:

	moviedb	title	genres	year	Adventure	Animation	Children	Comedy	Fantasy	Romance
0	1	Toy Story	[Adventure, Animation, Children, Comedy, Fantasy]	1995	1.0	1.0	1.0	1.0	1.0	0.0
1	2	Jumanji	[Adventure, Children, Fantasy]	1995	1.0	0.0	1.0	0.0	1.0	0.0
293	296	Pulp Fiction	[Comedy, Crime, Drama, Thriller]	1994	0.0	0.0	0.0	1.0	0.0	0.0
1246	1274	Akira	[Action, Adventure, Animation, Sci-Fi]	1988	1.0	1.0	0.0	0.0	0.0	0.0
1885	1968	Breakfast Club, The	[Comedy, Drama]	1985	0.0	0.0	0.0	1.0	0.0	0.0

5 rows × 24 columns

We'll only need the actual genre table, so let's clean this up a bit by resetting the index and dropping

the movieId, title, genres and year columns.

```
In [12]: #Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Dropping unnecessary issues due to save memory and to avoid issues
userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('
```

Out[12]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	IMAX	Documentary	War	Musical	Western	Film-Noir	(no genres listed)
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

Now we're ready to start learning the input's preferences!

To do this, we're going to turn each genre into weights. We can do this by using the input's reviews and multiplying them into the input's genre table and then summing up the resulting table by column. This operation is actually a dot product between a matrix and a vector, so we can simply accomplish by calling Pandas's "dot" function.

```
In [13]: inputMovies['rating']
```

```
Out[13]: 0    3.5
1    2.0
2    5.0
3    4.5
4    5.0
Name: rating, dtype: float64
```

```
In [14]: #Dot product to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
```

```
Out[14]: Adventure          10.0
Animation           8.0
Children            5.5
Comedy              13.5
Fantasy             5.5
Romance             0.0
Drama               10.0
Action               4.5
Crime                5.0
Thriller             5.0
Horror               0.0
Mystery              0.0
Sci-Fi               4.5
IMAX                 0.0
Documentary          0.0
War                  0.0
Musical               0.0
Western               0.0
Film-Noir             0.0
(no genres listed)   0.0
dtype: float64
```

Now, we have the weights for every of the user's preferences. This is known as the User Profile. Using this, we can recommend movies that satisfy the user's preferences.

Let's start by extracting the genre table from the original dataframe:

In [15]:

```
#Now let's get the genres of every movie in our original dataframe
genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1).drop('year')
genreTable.head()
```

Out[15]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Ho
movieId											
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [16]:

```
genreTable.shape
```

Out[16]:

```
(34208, 20)
```

With the input's profile and the complete list of movies and their genres in hand, we're going to take the weighted average of every movie based on the input profile and recommend the top twenty movies that most satisfy it.

In [17]:

```
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df.head()
```

Out[17]:

```
movieId
1      0.594406
2      0.293706
3      0.188811
4      0.328671
5      0.188811
dtype: float64
```

In [18]:

```
#Sort our recommendations in descending order
recommendationTable_df = recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df.head()
```

Out[18]:

```
movieId
5018      0.748252
26093     0.734266
27344     0.720280
148775    0.685315
6902      0.678322
dtype: float64
```

Now here's the recommendation table!

In [19]:

```
#The final recommendation table
```

```
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(20).keys())]
```

Out[19]:

	movieId		title	genres	year
664	673		Space Jam	[Adventure, Animation, Children, Comedy, Fantasy, Sci-Fi]	1996
1824	1907		Mulan	[Adventure, Animation, Children, Comedy, Drama, Romance]	1998
2902	2987		Who Framed Roger Rabbit?	[Adventure, Animation, Children, Comedy, Crime, Thriller]	1988
4923	5018		Motorama	[Adventure, Comedy, Crime, Drama, Fantasy, Mystery]	1991
6793	6902		Interstate 60	[Adventure, Comedy, Drama, Fantasy, Mystery, Sci-Fi]	2002
8605	26093		The Wonderful World of the Brothers Grimm, The	[Adventure, Animation, Children, Comedy, Drama, Family]	1962
8783	26340		The Twelve Tasks of Asterix, The (Les douze travaux d'Asterix)	[Action, Adventure, Animation, Children, Comedy, Family]	1976
9296	27344		Revolutionary Girl Utena: Adolescence of Utena	[Action, Adventure, Animation, Comedy, Drama, Romance]	1999
9825	32031		Robots	[Adventure, Animation, Children, Comedy, Fantasy]	2005
11716	51632		Atlantis: Milo's Return	[Action, Adventure, Animation, Children, Comedy, Family]	2003
11751	51939		TMNT (Teenage Mutant Ninja Turtles)	[Action, Adventure, Animation, Children, Comedy, Family]	2007
13250	64645		The Wrecking Crew	[Action, Adventure, Comedy, Crime, Drama, Thriller]	1968
16055	81132		Rubber	[Action, Adventure, Comedy, Crime, Drama, Thriller]	2010
18312	91335		The Gruffalo, The	[Adventure, Animation, Children, Comedy, Drama]	2009
22778	108540		Ernest & Célestine (Ernest et Célestine)	[Adventure, Animation, Children, Comedy, Drama]	2012
22881	108932		The Lego Movie	[Action, Adventure, Animation, Children, Comedy, Family]	2014
25218	117646		Dragonheart 2: A New Beginning	[Action, Adventure, Comedy, Drama, Fantasy, Thriller]	2000
26442	122787		The 39 Steps	[Action, Adventure, Comedy, Crime, Drama, Thriller]	1959
32854	146305		Princes and Princesses	[Animation, Children, Comedy, Drama, Fantasy]	2000
33509	148775		Wizards of Waverly Place: The Movie	[Adventure, Children, Comedy, Drama, Fantasy]	2009

Advantages and Disadvantages of Content-Based Filtering

Advantages

- Learns user's preferences
- Highly personalized for the user

Disadvantages

- Doesn't take into account what others think of the item, so low quality item recommendations might happen
- Extracting data is not always intuitive
- Determining what characteristics of the item the user dislikes or likes is not always obvious

Практическая работа №5

Извлечение и визуализация данных из БД

Цель занятия: приобрести практические навыки в области создания интеллектуальных систем с использованием современных средств разработки.

Пояснения к работе

В данной работе взаимодействие источниками данных будет проходить в Jupyter Notebook на базе Google Colab (см. Методические указания к Практической работе №3, <https://colab.research.google.com/notebooks/intro.ipynb>, <https://github.com/deepmipt/dlschl/wiki/Инструкция-по-работе-с-Google-Colab>).

Эта лабораторная работа в значительной степени полагается на библиотеки pandas и NumPy для обработки, анализа и визуализации данных. Основная библиотека для построения графиков, которую мы исследуем в этой лабораторной работе - это Folium.

Folium - это мощная библиотека Python, которая помогает создавать несколько типов карт. Тот факт, что результаты Folium интерактивны, делает эту библиотеку очень полезной для создания информационных панелей. Folium опирается на сильные стороны экосистемы Python и возможности сопоставления библиотеки Leaflet.js. Folium позволяет легко визуализировать данные, обработанные в Python, на интерактивной карте Leaflet. Библиотека имеет ряд встроенных наборов фрагментов из OpenStreetMap, Mapbox и Stamen и поддерживает настраиваемые наборы фрагментов с ключами API Mapbox или Cloudmade. Folium поддерживает наложения как GeoJSON, так и TopoJSON, а также привязку данных к этим наложениям для создания карт с цветовыми схемами.

Наборы данных:

Инциденты полицейского управления Сан-Франциско за 2016 год - инциденты полицейского управления из портала публичных данных Сан-Франциско. Инциденты получены из системы сообщений о преступлениях Департамента полиции Сан-Франциско (SFPD). Обновляется ежедневно, отображая данные за весь 2016 год.

Программа работы

1. Переходим к созданному в рамках П.р. №3 аккаунту в Google Colab и создаем там новый блокнот.
2. Импортируем необходимые для работы библиотеки:

```
import numpy as np # useful for many scientific computing in Python
import pandas as pd # primary data structure library
```

2. Устанавливаем библиотеку folium:

```
!pip install folium==0.5.0
import folium

print('Folium installed and imported!')
```

3. Создаем карты мира в Folium. Создается объект Folium Map и затем запускается на отображение. Карты интерактивны, поэтому вы можете увеличивать любую интересующую область, несмотря на начальный уровень масштабирования.

```
# define the world map
world_map = folium.Map()

# display world map
world_map
```

4. Создаем карту с центром в Канаде и изменяем уровень масштабирования, чтобы увидеть, как он влияет на визуализированную карту.

```
# define the world map centered around Canada with a low zoom level
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4)

# display world map
world_map
```

5. Еще одна интересная особенность Folium - вы можете создавать карты разных стилей. Попробуйте различные варианты и выберите тот, который по Вашему мнению лучше всего.

```
# create a Stamen Toner map of the world centered around Canada
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4, tiles='Stamen Toner')

# display map
world_map
```

```
# create a Stamen Toner map of the world centered around Canada
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4, tiles='Stamen Terrain')

# display map
world_map
```

```
# create a world map with a Mapbox Bright style.
world_map = folium.Map(tiles='Mapbox Bright')

# display the map
world_map
```

6. Загружаем датасет и запрашиваем обзор его содержимого.

```
df_incidents = pd.read_csv(  
    'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-  
data/CognitiveClass/DV0101EN/labs/Data_Files/Police_Department_Incidents_-_Previous_Year__2016_.csv')  
  
print('Dataset downloaded and read into a pandas dataframe!')  
df_incidents.head()
```

7. Объем датасета слишком большим, поэтому ограничиваем его до первых 100 записей.

```
# get the first 100 crimes in the df_incidents dataframe  
limit = 100  
df_incidents = df_incidents.iloc[0:limit, :]
```

8. Визуализируем, где преступления имели место в городе Сан-Франциско. Для примера использован стиль по умолчанию и начальный уровень масштабирования до 12 (студенту необходимо выбрать эти параметры самостоятельно).

```
# San Francisco latitude and longitude values  
latitude = 37.77  
longitude = -122.42  
# create map and display it  
sanfran_map = folium.Map(location=[latitude, longitude], zoom_start=12)  
  
incidents = folium.map.FeatureGroup()  
  
# loop through the 100 crimes and add each to the incidents feature group  
for lat, lng, in zip(df_incidents.Y, df_incidents.X):  
    incidents.add_child(  
        folium.features.CircleMarker(  
            [lat, lng],  
            radius=5, # define how big you want the circle markers to be  
            color='yellow',  
            fill=True,  
            fill_color='blue',
```

```
        fill_opacity=0.6
    )
)

# add incidents to map
sanfran_map.add_child(incidents)
```

9. Измените вид карты и представьте результаты в отчете.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое Folium?

Практическая работа №6

Передача данных с использованием протокола Ethernet

Цель занятия: получить навыки передачи данных по сети Ethernet с использованием языка программирования Python.

Пояснения к работе

Ethernet основан на протоколе TCP/IP.

Применяемая в IP-сетях архитектура клиент-сервер использует IP-пакеты для коммуникации между клиентом и сервером. Клиент отправляет запрос серверу, на который тот отвечает. В случае с TCP/IP между клиентом и сервером устанавливается соединение (обычно с двусторонней передачей данных), а в случае с UDP/IP - клиент и сервер обмениваются пакетами (дейтаграммами) с негарантированной доставкой. Каждый сетевой интерфейс IP-сети имеет уникальный в этой сети адрес (IP-адрес). Упрощенно можно считать, что каждый компьютер в сети Интернет имеет собственный IP-адрес. При этом в рамках одного сетевого интерфейса может быть несколько сетевых портов. Для установления сетевого соединения приложение клиента должно выбрать свободный порт и установить соединение с серверным приложением, которое слушает (listen) порт с определенным номером на удаленном сетевом интерфейсе. Пара IP-адрес и порт характеризуют сокет («гнездо») - начальную (конечную) точку сетевой коммуникации. Для создания соединения TCP/IP необходимо два сокета: один на локальной машине, а другой - на удаленной. Таким образом, каждое сетевое соединение имеет IP-адрес и порт на локальной машине, а также IP-адрес и порт на удаленной машине. Модуль `socket` обеспечивает возможность работать с сокетами из Python. Сокеты используют транспортный уровень согласно семиуровневой модели OSI (Open Systems Interconnection, взаимодействие открытых систем), то есть относятся к более низкому уровню, чем большинство описываемых в этом разделе протоколов. Уровни модели OSI:

Физический

Поток битов, передаваемых по физической линии. Определяет параметры физической линии.

Канальный (Ethernet, PPP, ATM и т.п.)

Кодирует и декодирует данные в виде потока битов, справляясь с ошибками, возникающими на физическом уровне в пределах физически единой сети.

Сетевой (IP)

Маршрутизирует информационные пакеты от узла к узлу.

Транспортный (TCP, UDP и т.п.)

Обеспечивает прозрачную передачу данных между двумя точками соединения.

Сеансовый

Управляет сеансом соединения между участниками сети.

Начинает, координирует и завершает соединения.

Представления

Обеспечивает независимость данных от формы их представления путем преобразования форматов. На этом уровне может выполняться прозрачное (с точки зрения вышележащего уровня) шифрование и дешифрование данных.

Приложений (HTTP, FTP, SMTP, NNTP, POP3, IMAP и т.д.)

Поддерживает конкретные сетевые приложения. Протокол зависит от типа сервиса.

Каждый сокет относится к одному из коммуникационных доменов. Модуль `socket` поддерживает домены UNIX и Internet. Каждый домен подразумевает свое семейство протоколов и адресацию. Данное изложение будет затрагивать только домен Internet, а именно протоколы TCP/IP и UDP/IP, поэтому для указания коммуникационного домена при создании сокета будет указываться константа `socket.AF_INET`.

В качестве задачи рассмотрим простейшую клиент-серверную пару. Сервер будет принимать строку и отвечать клиенту. Сетевое устройство иногда называют хостом (`host`), поэтому будет употребляться этот термин по отношению к компьютеру, на котором работает сетевое приложение.

Работа выполняется в среде Jupyter Notebook из пакета Anaconda (см. Практическая работа №3).

1. Подготовительная часть.

1.1. Создайте два отдельных Jupyter Notebook – блокнота в рабочем каталоге.

2. В первом блокноте наберите программу сервера, принимающего сообщения.

```

import socket, string
def do_something(x):
    lst = x[::-1]
    return lst
HOST = "" # localhost
PORT = 33333
srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srv.bind((HOST, PORT))
idx = 0
while idx<5:
    print ('Слушаю порт 33333')
    srv.listen(1)
    sock, addr = srv.accept()
    while 1:
        pal = sock.recv(1024)
        if not pal:
            break
        else:
            idx+=1
        print ('Получено от %s:%s:' % addr, pal.decode('utf-8'))
        lap = do_something(pal.decode('utf-8'))
        print ('Отправлено %s:%s:' % addr, lap)
        sock.send(lap.encode('utf-8'))
    sock.close()
print('Получено 5 сообщений')

```

3. Во втором блокноте наберите программу клиента, отправляющего сообщения.

```

import socket
HOST = "# удаленный компьютер (localhost)"
PORT = 33333 # порт на удаленном компьютере
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((HOST, PORT))
sock.send('фывыссыв'.encode('utf-8'))
result = sock.recv(1024)
sock.close()
print ('Получено:', result.decode('utf-8'))

```

4. Запустите программу сервера.

5. Запустите программу клиента. Проанализируйте результаты работы обеих программ.

6. Измените программу таким образом, чтобы клиенту вернулось сообщение «привет».

СПИСОК ЛИТЕРАТУРЫ

1. Теняков Е. И. Общие требования и правила оформления текстовых документов в учебном процессе : учеб.-метод. пособие / ЮРГТУ(НПИ). - Новочеркасск: Изд-во ЮРГТУ(НПИ), 2012. - 84 с.
2. Теняков Е. И. Общие требования и правила выполнения электрических схем, схем алгоритмов, программ, данных и систем: учеб.-метод. пособие / ЮРГТУ(НПИ). - Новочеркасск: Изд-во ЮРГТУ (НПИ), 2013. - 159 с.

Учебно-методическое издание
Шайхутдинов Данил Вадимович
Наракидзе Нури Дазмирович

Интеллектуальные системы и технологии

Издается в авторской редакции
Подписано в печать ___.2016. Формат 60×84 1/16 .
Бумага газетная. Ризография. Усл. печ. л. 1,56.
Уч.-изд. л.1,67. Тираж 50 экз. Заказ № _____

Южно-Российский государственный политехнический университет
(НПИ) им. М.И. Платова
Редакционно-издательский отдел ЮРГПУ (НПИ)
346430, г. Новочеркасск, ул. Просвещения, 132

Отпечатано в Издательско-полиграфическом комплексе «Колорит»
346430, г. Новочеркасск, пр. Платовский, 82Е,
тел.: 8(8635)226-442, 8-952-603-0-609,
e-mail: center-op@mail.ru