D3.js (или просто D3) это JavaScript-библиотека для обработки и визуализации данных. Она предоставляет удобные утилиты для обработки и загрузки массивов данных и создания DOM-элементов. D3 реализует подход, называемый fluent interface. При чтении кода он выглядит как цепочка методов. Каждый метод вызывается на объекте, который вернул предыдущий метод. Чтобы код было удобно читать, каждый вызов располагается на отдельной строчке.

Опишем работу основных модули, которые предоставляет d3.js и пример применения данной библиотеки для визуализации данных.
Пример взят с сайта http://bl.ocks.org и визуализирует связи между персонажами романа “Отверженные”

Для начала работы с d3.js подключим библиотеку добавив:
```
<script src="https://d3js.org/d3.v4.js"></script>
```

Теперь рассмотрим основные модули и функции, которые предоставляет d3.js для чтения обработки и визуализации данных

- d3.**select**(*selector*), d3.**selectAll**(*selector*) или d3.**selectAll**(*nodes*)

Данные методы ищут элементы в документе и создают выборки— обёртки набора элементов. Благодаря этому у нас есть доступ к методам библиотеки для модификации выбранных элементов.
Пример из index.html:
```
svg = d3.select("svg")
```

Далее рассмотрим основные методы для работы с содержимым выборки.

- selection.**attr**(*name*[, *value*])

Если value не указано, то данный метод возвращает значение указанного атрибута. Если value указано, то метод устаналвивает значение равное value.
Пример из index.html
```
width = +svg.attr("width"),
height = +svg.attr("height")
```

- selection.**style**(*name*[, *value*[, *priority*]])

Данный метод, устанавливает значения стиля элемента равным value, таким образом данный метод позволяет изменять внешний вид.

- selection.**append**(*name*)

Добавляет новый элемент в выборку
Пример из index.html
```
var node = svg.append("g")
```

- selection.**text**([*value*])

В зависимости от того, установлено ли значение value либо возвращает значение text, либо устанавливает его равному value.
Пример из index.html
```
node.append("title")
   .text(function(d) { return d.id; });
```

- selection.**data**([*values*[, *key*]])

Метод связывает выборку с массивом данных, порождая выборки enter и exit
Пример из index.html

```
var link = svg.append("g")
    .attr("class", "links")
  .selectAll("line")
  .data(graph.links)
  .enter().append("line")
```

Также данная библиотека предоставляет методы для работы с массивами данных.

- d3.**min**(array)
  возвращает минимальное значение в массиве data
- d3.**max**(array)
  возвращает максимальное значение
- d3.**sum**(array)
  возвращает сумму всех элементов массива
- d3.**median**(array)
  возвращает медиану массива
- d3.**mean**(array)
  получает среднее значение
- d3.**ascending**(array)/ d3.**descending(**array)
  представляют функции, упорядочивающие массив по возрастанию / по убыванию
- d3.**bisect**(array)
  функция, которая возвращает позицию в отсортированном массиве, куда можно поместить определенный объект, не нарушая порядка возрастания.
- d3.**nest**(array)
  представляет функцию, которая преобразует некоторый массив объект в другую иерархическую структуру, более удобную для визуализации данных

Для сопоставления данных d3.js предоставляет модуль d3.scales

- d3.**scale.linear**()

строит линейную шкалу

- d3**.scale.log()**

использует логарифмическую шкалу с основанием по умолчанию 10

- d3.**time.scale**()

позволяет сопоставлять временные отрезки и линейные интервалы.

Теперь рассмотрим основные способы визуализации, на основе наших данных.
D3.js использует векторную графику для визуализации. Применение SVG позволяет легко рисовать простейшие графические примитивы и затем из них складывать более сложные фигуры. Отличительной особенностью SVG является то, что эта технология позволяет применять стили CSS для настройки визуализации фигур, что дает нам дополнительный контроль над визуализацией. Рассмотрим основные объекты, которые можно создавать.

Прямая

- d3**.line**()

создает новый генератор с настройками по умолчанию

- line(data)

создает прямую для данного массива данных

line.**x**([x]) и line.y([**y**])
позваляют построить прямую на основе массива данных, если массив не является массивом пар чисел

Кривая

- d3.**curveBasis**(*context*)
- d3.**curveNatural**
- d3.**curveLinear**
- d3.**curveBundle**(*context*)
- d3.**curveCardinal**(*context*)
- d3.**curveStep**
- …

данные функции строят кривую, приближенную к заданным точкам, вид кривой будет зависеть, от того какой из этих методов будет выбран

Полярные координаты

- d3.**radialLine**()

содержит функции аналогичные line, только вместо ПДСК у нас полярная система координат.

Площади

- d3.**area**(data)

создает генератор для построения двумерных объектов, на основе массива данных, содержит функции похожие на функции d3.line, также как ПДСК, так и поддерживает полярные координаты.

- d3.**symbo**l()

позваляет создавать различные фигуры, есть различные методы, позваляющие определять характеристики фигур

- *symbol*.**type**([*type*])
- *symbol*.**size**([*size*]
- …

Теперь, рассмотрим более сложные структуры для визуализации: графы, модели, кластеры и деревья

force simulation(моделирование связей). Визуализируем связи между объектами, их влияние друг на друга (силы воздействия)

- d3.**forceSimulation**([*nodes*])

создает новую модель на вершинах nodes, но без связей. Моделирование начинается автоматически, но им можно управлять с помощью:

- *simulation*.**restart**()
- *simulation*.**stop**()

- *simulation*.**force**(*name*[, *force*])

Устаналивает связи между вершинами

- *simulation*.**find**(*x*, *y*[, *radius*])

ищет ближайшую вершину

Также вершинами можно управлять с помощью функций, моделирующих определенные процессы, примерами таких функций являются:

- d3.**forceLink**([*links*])
  Устанавливает связь между двумя вершинами
- d3.**forceCollide**([*radius*])
  Предотвращает столкновение вершин, представленных в виде окружностей с определенным радиусом
- d3.**forceCenter**([*x*, *y*])
  Стягивает вершины к центру (указанной точке)
- …

Особенно для нас важна forceLink так она позваляет устанавливать связи между вершинами модели рассмотрим ее методы.
link.**links**([*links*])
Данная функции свзи на основе переданного в нее массива связей.
link.**id**([*id*])
устанавливает id в указанное значение
link.**strength**([*strength*])
link.**distance**([*distance*])
действуют аналгочино link.id для strength и distance соответсвенно.

Пример forceSimulation из index.html

```
var simulation = d3.forceSimulation()
   .force("link", d3.forceLink().id(function(d) { return d.id; }))
   .force("charge", d3.forceManyBody())
   .force("center", d3.forceCenter(width / 2, height / 2));
```

Различные иерархические структуры

- d3.**hierarchy**(*data*[, *children*])

конструктор на основе массива данных
У иерархии есть различные методы для построения и работы с элементами
Данные можно визуализировать различными способами, например построить дерево, кластер или набор

- d3.**cluster**()
- d3.**tree**()
- d3.**pack**()
- …

Вызывают конструкторы для визуализации иерархии выбранным способом.

Библиотека d3.js дает возможность выполнять анимированные переходы для выборок, используя selection.transition()

- selection.**transition**([name])

выполняет переход для selection()

Используя различные функции этого модуля можно настроить, что будет изменяться и каким образом. Пример:

- transition.style

плавный переход к модернизированному значению свойства.

- transition.tween

задать пользовательские анимации оператору для их запуска составе перехода.

- transition.filter

фильтр перехода, основанный на значениях data.

- transition.transition

когда этот переход закончится, запустите другой на тех же элементах.

Также в d3.js есть модули geometry и geography для работы с соответсвующими данными и их визуализации.