

[사용자] React-financial-charts

| | |
|-------------|--------------------------------------|
| ☰ 구현 어려움 정도 | ★★★★ |
| ☑ 구현 완료 | ☑ |
| 📅 기간 | @December 6, 2024 → December 7, 2024 |
| ☑ 정리 완료 | ☑ |

Webpack App

<https://react-financial.github.io/react-financial-charts/?path=/story/intro--page>

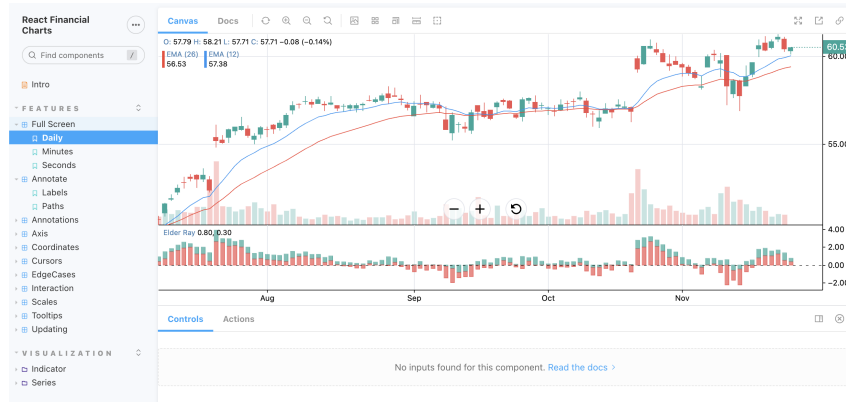


차트 라이브러리 중, 캔들 차트를 만들 수 있는 라이브러리로 react-financial-charts라는 라이브러리를 찾게 되었습니다. 사용법을 확인하기 위해 공식 docs를 확인해보려 했으나 공식 문서가 거의 없었고, github과 code snadbox, storybook을 참고하면서 사용법을 익혀야 했습니다.

react-financial-charts 설치

```
pnpm install react-financial-charts d3-format d3-time-format
```

- react-financial-charts 뿐만 아니라, format에 필요한 라이브러리도 함께 install 합니다.
- React 사용자라면 설치만 하고 차트를 만들러 넘어가면 되지만, Next.js 사용자라면 `next.config.js`의 `reactStrictMode`를 `false`로 바꿔야합니다.

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: false,
  swcMinify: true,
}

export default nextConfig;
```

- 해당 세팅을 하지 않으면 캔들 차트에 마우스 호버링 이벤트가 발생하지 않으며, 패닝 시 차트가 사라지는 오류가 발생합니다.
- react-financial-charts는 SSR 환경에서 제대로 동작하지 않기 때문에 Next.js에서 react-financial-charts로 만든 차트를 컴포넌트에 import할 때 아래의 예시와 같이 `dynamic import`하기를 권장합니다.

```
const CandleChart = dynamic(() => import("components/charts/CandleChart"), {
  ssr: false,
});
```

react-financial-charts 주요 기능

react-financial-charts를 활용해 차트를 만들 때 주로 import하는 기능들에 대해 각각 간략하게 설명하겠습니다.

```
import { format } from "d3-format";
import { timeFormat } from "d3-time-format";
import {
  elderRay,
  ema,
  discontinuousTimeScaleProviderBuilder,
  Chart,
  ChartCanvas,
  CurrentCoordinate,
  BarSeries,
  CandlestickSeries,
  ElderRaySeries,
  LineSeries,
  MovingAverageTooltip,
  OHLCTooltip,
  SingleValueTooltip,
  lastVisibleItemBasedZoomAnchor,
  XAxis,
  YAxis,
  CrossHairCursor,
  EdgeIndicator,
  MouseCoordinateX,
  MouseCoordinateY,
  ZoomButtons,
} from "react-financial-charts";
```

1. d3 format

- `format`: 숫자를 원하는 형식으로 포맷팅하는 함수입니다.
- `timeFormat`: 시간 및 날짜를 원하는 형식으로 포맷팅하는 함수입니다.

2. react-financial-charts

- `discontinuousTimeScaleProviderBuilder`: 시간 축의 불연속적인 데이터에 적합한 스케일을 제공하는 빌더입니다.

금융 데이터는 주말이나 공휴일에 거래가 없기 때문에 연속적인 데이터가 아닙니다. 이럴 때, `discontinuousTimeScaleProviderBuilder`를 사용하면 이러한 비연속적인 시간 데이터를 적절하게 처리해 차트에 표시할 수 있습니다.

아래 코드 예시와 같이 `ScaleProvider`를 생성하고 유용한 함수를 반환합니다.

```
const ScaleProvider = discontinuousTimeScaleProviderBuilder().inputDateAccessor(
  (d) => new Date(d.date)
);

const { data, xScale, xAccessor, displayXAccessor } = ScaleProvider(
  initialData
);
```

- `data`: 처리된 데이터 배열. (주말이나 공휴일 등 거래가 없는 날은 제외된 데이터)
- `xScale`: 비연속적인 시간 데이터를 처리한 x축 스케일
- `xAccessor`: x축 데이터를 어떻게 가져올지 정의하는 함수 `(d) => new Date(d.date)`
- `displayXAccessor`: 차트의 x축에 실제로 어떤 값을 표시할지 정의하는 함수
- `ChartCanvas`: 전체 차트의 캔버스. 모든 차트 컴포넌트는 이 안에 위치하므로, 컨테이너와 유사 기능을 합니다.

```

<ChartCanvas
  height={height}
  ratio={3}
  width={width}
  margin={margin}
  data={data}
  displayXAccessor={displayXAccessor}
  seriesName="Data"
  xScale={xScale}
  xAccessor={xAccessor}
  xExtents={xExtents}
  zoomAnchor={lastVisibleItemBasedZoomAnchor}
>

```

- 기본적인 container style 세팅, 차트에 활용되는 데이터를 설정합니다.
- `discontinuousTimeScaleProviderBuilder` 에서 반환된 함수들을 할당할 수 있습니다.
- `displayXAccessor` 로 x축에 실제로 표시될 데이터(주로 시계열 날짜, 시간 데이터)를 반환합니다.
- `xAccessor` 로 x축에 표시될 데이터(주로 시계열 날짜, 시간 데이터)를 어떻게 가져올지 결정합니다.
- `seriesName` 으로 여러 Chart 컴포넌트가 하나의 ChartCanvas안에 있을 때 각 시리즈를 구분하기 위해 사용됩니다.
- `xScale` 은 x축의 스케일을 정의합니다. 금융 시계열 데이터가 화면에 어떻게 표시될지 결정합니다.
- `xExtents` 는 x축의 최소 및 최대 범위를 정의합니다. 이 속성은 배열로 제공되며, 데이터의 초기 표시 범위를 지정하는데 사용됩니다.

```

const max = xAccessor(data[data.length - 1]);
const min = xAccessor(data[Math.max(0, data.length - 100)]);
const xExtents = [min, max + 5];

```

7. `Chart`: 서로 다른 차트들을 묶어 주는데 사용하는 차트 컴포넌트의 기본 래퍼입니다.

예를들어, 가격차트(candle chart + line chart)와 거래 대금 차트(bar chart)를 각각의 Chart로 감쌉니다.

`BarSeries`, `LineSeries`, `CandlestickSeries`, `ElderRaySeries` 등을 통해 차트의 종류를 정하고,

`yAccessor` 속성에 반환할 데이터를 할당합니다.

`CandlestickSeries`의 경우, `yAccessor`를 설정 안해도 react-financial-charts가 자동으로 `yAccessor`에 `open`, `high`, `low`, `close` 속성을 반환합니다.

아래는 예시 거래 대금 차트입니다.

```

const volumeSeries = (data) => {
  return data.volume;
};
...

<Chart
  id={2}
  height={barChartHeight}
  origin={barChartOrigin}
  yExtents={barChartExtents}
>
  <BarSeries fillStyle={volumeColor} yAccessor={volumeSeries} />
</Chart>

```

8. `XAxis`, `YAxis`: x축 컴포넌트와 y축 컴포넌트를 설정합니다.

```

<Chart id={3} height={chartHeight} yExtents={candleChartExtents}>
  <XAxis showGridLines showTickLabel={false} />
  <YAxis showGridLines tickFormat={pricesDisplayFormat} />

```

```
<CandlestickSeries />
</Chart>
```

- tickFormat에 d3-format을 활용하여 만든 함수를 할당하여 표시될 데이터의 format을 설정할 수 있습니다.

9. **CurrentCoordinate**: 현재의 마우스 위치나 최근의 데이터 포인트에서 특정 지표의 값을 표시하는 역할을 합니다.

이를 통해 사용자는 차트에서 특정 지점의 정확한 값을 쉽게 확인할 수 있습니다.

```
<Chart ...>
<LineSeries yAccessor={ema26.accessor()} strokeStyle={ema26.stroke()} />
  <CurrentCoordinate
    yAccessor={ema26.accessor()}
    fillStyle={ema26.stroke()}
  />
...

```

위의 예시에서는 LineSeries를 통해 이동평균선 선 차트가 그려질 것이고, CurrentCoordinate 컴포넌트를 통해, 사용자가 차트 내에서 마우스로 이동할 때, 이동평균선에 현재 위치에 점이 표시되어 선을 따라 이동하게 됩니다.

10. **ema**: 이동 평균선(EMA)을 계산하는 함수입니다.

```
const ema12 = ema()
  .id(1)
  .options({ windowSize: 12 })
  .merge((d, c) => {
    d.ema12 = c;
  })
  .accessor((d) => d.ema12);

const ema26 = ema()
  .id(2)
  .options({ windowSize: 26 })
  .merge((d, c) => {
    d.ema26 = c;
  })
  .accessor((d) => d.ema26);
```

options의 windowSize를 통해 명시적으로 EMA 계산에 사용될 날짜 범위를 표기합니다.

merge는 계산된 EMA 값을 원래의 데이터 객체에 어떻게 통합할지 정의하는 함수를 제공합니다.

accessor 메서드를 통해 반환할 ema데이터의 종류를 반환합니다.

생성된 ema 변수는 컴포넌트의 yAccessor 등 속성에 활용됩니다.

```
<Chart ...>
  <LineSeries yAccessor={ema26.accessor()} strokeStyle={ema26.stroke()} />
</Chart>
```

11. **elderRay**: Elder Ray 인디케이터를 계산하는 함수

주로 주가의 불균형 (Bulls Power와 Bears Power)을 측정하는데 사용됩니다.

```
const elder = elderRay();
```

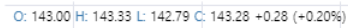
간단하게 호출하는 것만으로 elderRay 지표를 사용할 준비가 되는 이유는 이 함수가 기본 설정 값들을 사용해서 Elder Ray 지표 객체를 반환하기 때문입니다.

ema와 마찬가지로 options, merge 등의 메서드를 추가하여 원 데이터에 통합하는 등 설정할 수 있습니다.

아래와 같이 elderRay 차트를 만드는데 활용됩니다.

```
<Chart
  id={4}
  height={elderRayHeight}
  yExtents={[0, elder.accessor()]}
  origin={elderRayOrigin}
  padding={{ top: 8, bottom: 8 }}
>
  <ElderRaySeries yAccessor={elder.accessor()} />
  ...
</Chart>
```

12. `OHLCTooltip` : OHL(Open, High, Low, Close) 값을 표시하는 툴팁 컴포넌트



```
<OHLCTooltip origin={[8, 16]} />
```

origin을 통해 차트에 Tooltip이 표시될 위치를 지정합니다.

별 다른 속성을 주지 않아도 recat-financial-charts에서 자동으로 open, high, low, close, change 캔들 데이터에 대한 툴팁을 제공해 줍니다.

13. `MovingAverageTooltip` : 이동 평균선의 툴팁을 표시하는 컴포넌트입니다.



```
<MovingAverageTooltip
  origin={[8, 24]}
  options={[
    {
      yAccessor: ema26.accessor(),
      type: "EMA",
      stroke: ema26.stroke(),
      windowSize: ema26.options().windowSize
    },
    {
      yAccessor: ema12.accessor(),
      type: "EMA",
      stroke: ema12.stroke(),
      windowSize: ema12.options().windowSize
    }
  ]}
/>
```

origin을 통해 차트에 이동평균선 Tooltip이 표시될 위치를 지정합니다.

options를 통해 표시될 Tooltip을 상세하게 설정할 수 있습니다.

14. `SingleValueTooltip` : 특정 값에 대한 툴팁을 표시하는 컴포넌트입니다.



아래 예시를 통해 ElderRay에 대한 툴팁을 표시할 수 있습니다.

```

<SingleValueTooltip
  yAccessor={elder.accessor()}
  yLabel="Elder Ray"
  yDisplayFormat={(d) =>
    `${pricesDisplayFormat(d.bullPower)}, ${pricesDisplayFormat(
      d.bearPower
    )}`
  }
  origin={[8, 16]}
/>

```

15. `MouseCoordinateX`, `MouseCoordinateY` : 마우스의 x, y 좌표 값을 표시하는 컴포넌트입니다.



```

<MouseCoordinateX displayFormat={timeDisplayFormat} />
<MouseCoordinateY
  rectWidth={margin.right}
  displayFormat={pricesDisplayFormat}
/>

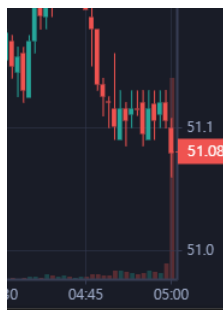
```

마우스가 이동하며 x축, y축에 표시될 값을 설정하는 컴포넌트입니다.

`displayFormat`으로 표시될 좌표 값의 format을 설정할 수 있습니다.

위의 예시에서 `MouseCoordinateX`로 인해 사용자가 차트 내에서 마우스를 이동하면 X축 영역에 해당하는 날짜가 X축 아래에 표시됩니다.

16. `EdgeIndicator` : 차트의 가장자리에 현재의 주요 값을 표시하는 컴포넌트입니다.



```

<EdgeIndicator
  itemType="last"
  rectWidth={margin.right}
  fill={openCloseColor}
  lineStroke={openCloseColor}
  displayFormat={pricesDisplayFormat}
  yAccessor={yEdgeIndicator}
/>

```

itemType에 last를 할당하여 마지막 데이터 값을 표시합니다.

17. `ZoomButtons` : 차트 확대/축소 버튼을 표시하는 컴포넌트입니다.



18. `CrossHairCursor` : 차트 내에서 교차선 커서를 표시하는 컴포넌트입니다.
19. `LastVisibleItemBasedZoomAnchor` : 마지막으로 보이는 아이템을 기준으로 줌 기능을 제공하는 앵커
20. `WithDeviceRatio` : 디바이스의 픽셀 비율에 따라 차트 렌더링을 최적화하는 Higher-Order-Component(HOC)
21. `WithSize` : 컴포넌트의 크기를 자동으로 조절하는 HOC

참고

<https://github.com/react-financial/react-financial-charts>

<https://codesandbox.io/s/react-financial-charts-demo-forked-96uyw?file=/src/index.js:0-5604>