Visualization checklist

Accessibility - Visualization checklist epic ticket: STP-1414 - Accessibility - Visualization checklist TO DO

Recommended changes per WCAG 2.1 AA compliance effort

Keyboard - WCAG A 2.1.1*

WG TODO: Investigate how to make tooltips accessible to keyboard users: (5-8 points)

- As a general rule, think about how to represent the data rather than how to solely visualize it. Describe it in words, provide data tables,
- We do need to enable keyboard access to our chart tooltips. Having them appear on hover only is insufficient. We should emulate what Highch arts or ReChart has done for their accessible tooltips.
    - This is a HIGH PRIORITY to meet WCAG 2.1 AA.
    - Before the charts, we need to add visible instructions for keyboard users explaining that they can tab to the chart below and then use the arrow keys to open and navigate between tooltips.
- We should continue with adding chart descriptions. It is ok that they are verbose and describe what is shown on the chart and its axes in exhaustive detail. We DO, however, need to add a Read More/Less ellipsis that reads ONLY the truncated copy to screen reader users, and then remove aria-hidden on the Read More button so they can choose when to hear the full description—just as mouse and keyboard users are able to do now.
- We should continue to add text tables to charts where those tables are of manageable size.
    - Manageable size: <20 rows, <10 columns, single row should represent a single data point.
    - You CAN use a subset of the data to represent its conclusions in a smaller, easier to understand table
    - Add extra filters to controls when viewing the text table to further filter the data and make the size more manageable
- If the table is too large or too complex to be understandable and can't be represented by a subset of data, then solely present a button for downloading the full data in a csv file.
- We should continue to offer the complete data shown on a chart as a csv download.

Other questions and future development opportunities

Make charts understandable:

- Ensure information complexity is appropriate, i.e., simply the presentation of data to increase understanding. Specifically, viz must not have more than one Y or X axis without an option to separate the data into two charts. Viz must not encode along a third spatial dimension (z axis) unless the data itself is 3D (sensory, modeling, etc). Viz should not contain more than 5 data categories. (closest is WCAG AAA 3.1.5 Only level A and AA are required, level AAA is optional.)

New controls:

- User should be able to undo or redo their actions. We already have a reset button on many charts, perhaps that is sufficient? (closest is WCAG A 2.5.2)
- If patterns or fill textures are used, user must be able to turn them off.

Please note

Per the UW Accessibility team, if the chart has more than 20 data points, it's best to make it aria-hidden and add a text-only table as an option for screen reader users. If the chart has less than 20 data points, you can try to make the chart contents accessible via aria-labels as described in the Optional section below. It really depends on the type of data presented and how easy it is to understand the data via the aria-labels.

Required

1.Chart Descriptions (WCAG A 1.1.1)*

WG TODO: Viz team to discuss and understand what HighCharts does out of the box for their chart descriptions

- The UW Accessibility team recommends creating **visible** chart descriptions that include information about what is shown, including data highlights, color info., axis information, extra information that might be noted via an * in the chart. The idea being that everyone could benefit from some language that interprets the chart. These chart descriptions might also help screen reader users discuss the charts with sighted users as well as orient screen users to the.
    - Useful data highlights include: the max value, min value, and possibly median value (where possible) from the data points
    - Simple sentences with punctuation to allow for pause are best.
    - Give a general description of the chart first, then give specific data points
    - If there is room on the viz, a list format is helpful when traversing back through the information via screen reader as well as visually but is not required
- Chart descriptions should be dynamic and update when the chart updates
- **A confirmed good example from the UW Accessibility team**: "A line chart showing the population trend over time is shown in orange. The X axis displays years. The Y axis displays population. Starting in 1990, the population was 48.6k. Ending in 2100, the population is expected to be 109.0k. The population trend is increasing.
- In Country Profiles, we have added these descriptions as collapsed text that can be expanded. The code below renders a chart description that reads, "A line chart showing this location's fertility rate (shown in purple) trending down from 2.1 in 1990 to 1.8 in 2017 to a projected value of 1.5 in 2100. For comparison, the High-income region's fertility rate (shown in yellow) was 2.1 in 1990 and is expected to be 1.5 in 2100, and the Global fertility rate (shown in green) was 3.1 in 1990 and is expected to be 1.7 in 2100. The X axis shows year from 1990 to 2100. The Y axis shows total fertility rate from 0.6 to 3.5."

Visible chart descriptions

```
// Because of how Country Profiles is set up, we need to create
// a <div> to hold the description in index.php and then render
// the <ChartDescription/> component into it via React.

// index.php
 <div id="forecasted_fertility_block" class="block">
    <h2 class="app-string" data-app-string="195" id="forecasted_ferility_title"></h2> // chart
title
    <div class="visually-hidden" id="forecasted-fertility-line-summary"></div> // target DOM
element for chart description
    <figure id="forecasted-fertility-line" class="country-profile chart"></figure> // the chart
will be rendered into this figure
    <p> // Information about the data used in the chart
                <span class="visually-hidden app-string" data-app-string="2"></span>
              <span class="app-string" data-app-string="197"></span>
    </p>
    <p class="app-string-container"> // Link to the article where the data was published.
      <a target="_blank" rel="noopener noreferrer" href="https://doi.org/10.1016/S0140-6736(20)
30677-2">
        <span class="app-string" data-app-string="204"></span>
        <span><?php print($DOI_TITLES['https://doi.org/10.1016/S0140-6736(20)30677-2']); ?><
/span>
      </a>
    </p>
</div>

// ForecastedFertility.js
// Use the getAriaLabel function to create the chart description.
function getAriaLabel(locations) {
      const [
        countryObserved,
        countryExpected,
        regionObserved,
        regionExpected,
        globalObserved,
        globalExpected,
      ] = data.map((dataset) => dataset.map((d) => d.mean.toFixed(1)));

      const countryValues = generateCell(countryObserved, countryExpected);
      const regionValues = generateCell(regionObserved, regionExpected);
      const globalValues = generateCell(globalObserved, globalExpected);

      const countryTrend = (countryValues[0] > countryValues[2]) ? 'down' : 'up';

      const domainBounds = getLineDomain(chartData, axisLabels);
      const codomainBounds = getLineCodomain(chartData, minYScale);

      return `A line chart showing this location's fertility rate (shown in purple) trending
        ${countryTrend} from ${countryValues[0]} in ${startYear} to ${countryValues[1]}
        in ${currentYear} to a projected value of ${countryValues[2]} in ${endYear}.
```

```
        For comparison, the ${locations[1]} region's fertility rate (shown in yellow)
        was ${regionValues[0]} in ${startYear} and is expected to be ${regionValues[2]}
        in ${endYear}, and the ${locations[2]} fertility rate (shown in green) was
        ${globalValues[0]} in ${startYear} and is expected to be ${globalValues[2]} in
${endYear}.
        The X axis shows ${axisLabels.x.text.toLowerCase()} from ${domainBounds.min} to
        ${domainBounds.max}. The Y axis shows ${axisLabels.y.text.toLowerCase()}
        from ${format(formatter)(codomainBounds.min).replace('G', 'B')}
        to ${format(formatter)(codomainBounds.max).replace('G', 'B')}.`;
    }

    // Render aria-label/chart description component.
    renderChartDescription(parent, getAriaLabel(localizedLocationNames));

// util.js
 renderChartDescription(parent, chartDescription) {
    const descriptionContainer = document.querySelector(`#${parent}-summary`);
    const descriptionRoot = createRoot(descriptionContainer);
    descriptionRoot.render(<ChartDescription description={chartDescription} />);
  },

// ChartDescription.jsx
import React, { useCallback, useState } from 'react';
import { Typography } from 'antd';
import PropTypes from 'prop-types';

export default function ChartDescription({ description }) {
  const { Paragraph } = Typography;
  const [ellipsis, setEllipsis] = useState(true);

  const handleClick = useCallback(() => {
    setEllipsis(!ellipsis);
  }, [ellipsis, setEllipsis]);

  return (
    <div className="ellipsis-container">
      <Paragraph
        ellipsis={ellipsis}
      >
        {description}
      </Paragraph>
      <button aria-hidden="true" className="ellipsis-button" type="button" onClick={handleClick}
>
        {ellipsis ? 'Read more' : 'Read less'}
      </button>
    </div>
  );
}

ChartDescription.propTypes = {
  description: PropTypes.string,
};

ChartDescription.defaultProps = {
  description: '',
};
```

- Using an aria-label to describe the chart is also an option, especially if you do not want a visible description.

Chart descriptions as aria-labels

```
// Create the aria label in the View component
function getAriaLabel(chartData) {
 const [populationObserved, populationForecasted] = chartData;
 const formattedStartMean = util.formatWithSuffixToPrecision(1, populationObserved[0].mean);
 const formattedCurrentMean = util.formatWithSuffixToPrecision(1,  populationObserved
[populationObserved.length - 1].mean);
 const formattedEndMean = util.formatWithSuffixToPrecision(1, populationForecasted
[populationForecasted.length - 1].mean);
 const trend = (populationObserved[0].mean > populationForecasted[populationForecasted.length -
1].mean) ? 'a decrease' : 'an increase';
         return `A line chart showing how the population will change. The overall trend from
    ${startYear} to ${endYear} shows ${trend} in population with ${formattedStartMean} in
    ${startYear}, ${formattedCurrentMean} in ${currentYear}, and is expected to be
    ${formattedEndMean} in ${endYear}.`;
}

// Pass the aria label to the chart component
ForecastedLinechart({
 ...
 chartAriaLabel: getAriaLabel(data),
 ...
});

// Use the aria-label in the chart component
function buildChart() {
 // create chart element
 chart = d3.select(`#${parentID}`).append('svg')
  .attr('class', `line-chart line-chart--${parentID}`)
  .attr('height', height)
  .attr('width', width)
  .attr('preserveAspectRatio', 'none')
  .attr('overflow', 'hidden')
  .attr('aria-label', chartAriaLabel); // add aria-label attribute
```

2.Text-only tables (WCAG A 1.1.1)*

WG TODO: Viz team to determine text table continuity between HighCharts PoC and the text table mock up for non-HighCharts charts

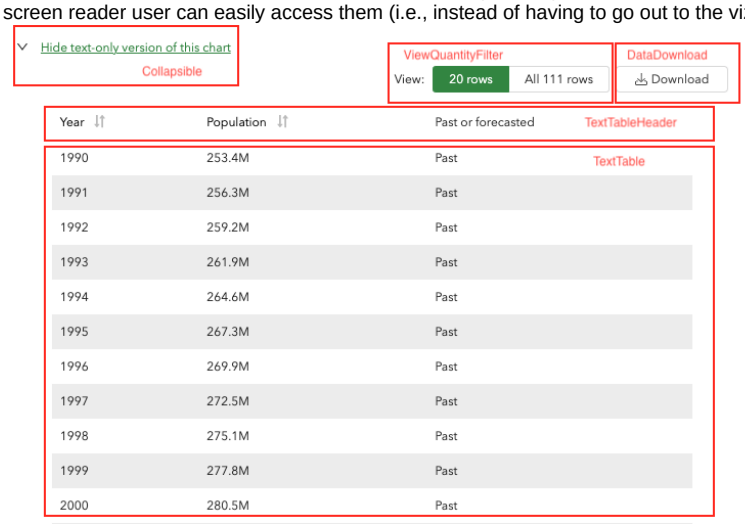
WG TODO: Viz team to research and ensure employment of the correct methods of tabbing through the text tables and reading out the data

- Wrap the chart (e.g., svg) in a `<figure>` element instead of a `<div>` . `<figure>` is a semantic element and provides helpful information to screen reader users.
- Add an aria-label to the chart's main `<svg>` that alerts screen reader users to the text-only version.

Aria-label to alert users to text-only table

```
chart = d3.select(`#${parentID}`).append('svg')
      .attr('class', `line-chart line-chart--${parentID}`)
      .attr('height', height)
      .attr('width', width)
      .attr('preserveAspectRatio', 'none')
      .attr('overflow', 'hidden')
      .attr('role', 'img')
      .attr(
        'aria-label',
        `As this chart contains many data points, you can view a
       text-only version via the link below.`,
      );
```

- Create text-only tables for each chart that include all data shown in the chart. Create simple accessible tables. Try to reduce complexity as much as possible (e.g., every column should have a heading even if the heading is repeated).
    - Manageable size: <20 rows, <10 columns, single row should represent a single data point.
    - You CAN use a subset of the data to represent its conclusions in a smaller, easier to understand table.
        - A snapshot of the data for the casual user would be best (this includes showing data points at evenly spaced intervals as is down in Country Profiles)
        - Add a data disclosure that the data is truncated and may not include important inflection point
    - Add extra filters to controls when viewing the text table to further filter data and make size more manageable
    - Country or state level data should be represented in one table
    - If the table is too large or too complex to be understandable and can't be represented by a subset of data, then solely present a button for downloading the full data in a csv file.
        - ie. County level and 5x5km raster data should be data download only
- Text table styling and examples of how to include the data table view option are in this Content and design text table figma file
    - See the split section for an example of how to add the data table control into the control panel
    - See components section to view styling of different text table components
- Country Profiles uses Antd's Collapsible to show/hide the tables. There is a ViewQuantityFilter component for tables with more than 20 rows. A Download button is included within the TextTable component (but can also be used outside of the tables) to allow data downloads where a screen reader user can easily access them (i.e., instead of having to go out to the vizhub template).



Collapsible - example from Country Profiles

```
import { Collapse } from 'antd';
import { noop } from 'lodash';
import PropTypes from 'prop-types';
import React, { useCallback, useState } from 'react';

const { Panel } = Collapse;
export default function Collapsible({ children, localize }) {
  const [isCollapsed, setIsCollapsed] = useState(true);
  const headerCollapsed = localize('appString', 231, 'View the text only version of this chart');
  const headerExpanded = localize('appString', 232, 'Hide the text only version of this chart');

  const onChange = useCallback(() => {
    setIsCollapsed(!isCollapsed);
  }, [isCollapsed]);

  const headerText = isCollapsed ? headerCollapsed : headerExpanded;

  return (
    <Collapse bordered={false} onChange={onChange}>
      <Panel header={headerText}>{children}</Panel>
    </Collapse>
  );
}

Collapsible.propTypes = {
  children: PropTypes.node,
  localize: PropTypes.func,
};

Collapsible.defaultProps = {
  children: null,
  localize: noop,
};
```

Format data for the text table - Example from Country Profiles Forecasted Population Chart

```
function formatForTextTable([populationObserved, populationForecasted]) {
    const formattedObservedData = populationObserved.map(({ mean, year }) => ({
      year,
      population: formatWithSuffixToPrecision(mean),
      pastOrForecasted: localize('appString', 205, 'Past'),
    }));
    const formattedForecastedData = populationForecasted.map(({ mean, year }) => ({
      year,
      population: formatWithSuffixToPrecision(mean),
      pastOrForecasted: localize('appString', 187, 'Forecasted'),
    }));
    return {
      // column header rows
      header: [
        {
          label: localize('appString', 94, 'Year'),
          name: 'year',
          sortable: true,
        },
        {
          label: localize('appString', 186, 'Population'),
          name: 'population',
          sortable: true,
        },
        {
          label: localize('appString', 236, 'Past or forecasted'),
          name: 'pastOrForecasted',
          sortable: false,
        },
      ],
      // table body rows
      body: [...formattedObservedData, ...formattedForecastedData],
    };
  }
```

TextTable - Example from Country Profiles

```
/* eslint-disable react/no-array-index-key */
/* eslint-disable react/forbid-prop-types */
/* eslint-disable react/jsx-props-no-spreading */
import PropTypes from 'prop-types';
import React, { useCallback, useMemo, useState } from 'react';
import { faSortAlt, faSortAmountUpAlt, faSortAmountDown } from '@fortawesome/pro-solid-svg-icons';
import { noop, orderBy } from 'lodash';

import { DEFAULT_TABLE_ROWS } from '../../constants';
import TextTableHeader from './TextTableHeader';
import ViewQuantityFilter from '../ViewQuantityFilter';
import DataDownload from '../DataDownload';

export default function TextTable({
  caption,
  columnToFormat,
  data,
  downloadData,
  filename,
  formatter,
  localize,
  parent,
  sourceFootnoteId,
  sourceUrl,
}) {
  const { header, body } = data;

  const [sortState, setSortState] = useState('none');
```

```
const [sortColumn, setSortColumn] = useState(undefined);
const [selectedQuantity, setSelectedQuantity] = useState(20);

// Sort the body rows based on sortState.
// Ensure number data are passed as typeof Number
// then sort will be by Number, not String.
const sortedBody = useMemo(() => {
  if (sortState === 'ascending') {
    return orderBy(body, [sortColumn], ['asc']);
  }
  if (sortState === 'descending') {
    return orderBy(body, [sortColumn], ['desc']);
  }
  return body;
}, [body, sortColumn, sortState]);

// Get the total number of data rows.
const totalRows = sortedBody.length;

// Get the options for the quantity filter.
const viewQuantityOptions = useMemo(
  () => [
    {
      label: `${DEFAULT_TABLE_ROWS} ${localize('appString', 234, 'rows')}`,
      value: DEFAULT_TABLE_ROWS,
    },
    {
      label: `${localize('appString', 235, 'All')} ${totalRows}
    ${localize('appString', 234, 'rows')}`,
      value: totalRows,
    },
  ],
  [localize, totalRows],
);

// Filter the sorted body rows down per the quantity filter.
const sortedAndFilteredBody = useMemo(
  () => sortedBody.slice(0, selectedQuantity),
  [sortedBody, selectedQuantity],
);

const handleSort = useCallback(
  (name) => {
    if (sortColumn === name) {
      // There are 3 states for sortState.
      if (sortState === 'none') {
        setSortState('ascending');
      } else if (sortState === 'ascending') {
        setSortState('descending');
      } else if (sortState === 'descending') {
        setSortState('none');
      }
    } else {
      setSortColumn(name);
      setSortState('ascending');
    }
  },
  [sortColumn, sortState, setSortColumn, setSortState],
);

const getSortIcon = useCallback(
  (name) => {
    if (sortColumn === name) {
      if (sortState === 'ascending') {
        return faSortAmountUpAlt;
      }
      if (sortState === 'descending') {
        return faSortAmountDown;
      }
    }
    return faSortAlt;
```

```
  },
  [sortColumn, sortState],
);

const getSortIconColor = useCallback(
  (name) => {
    if (sortColumn === name) {
      return sortState === 'none' ? '#bbbbbb' : '#3e853c';
    }
    return '#bbbbbb';
  },
  [sortColumn, sortState],
);

const tableRows = useMemo(
  () =>
    sortedAndFilteredBody.map((rowObject, index) => (
      <tr className="text-table-row" key={`${parent}-${index}`}>
        {Object.entries(rowObject).map(([keyProp, value]) => {
          if (keyProp === columnToFormat) {
            return <td key={`${keyProp}-${value}-${index}`}>{formatter(value)}</td>;
          }
          return <td key={`${keyProp}-${value}=${index}`}>{value}</td>;
        })}
      </tr>
    )),
  [columnToFormat, formatter, parent, sortedAndFilteredBody],
);

return (
  <>
    <div className="view-download-wrapper">
      {totalRows > DEFAULT_TABLE_ROWS && (
        <ViewQuantityFilter
          parent={parent}
          selectedQuantity={selectedQuantity}
          setSelectedQuantity={setSelectedQuantity}
          viewQuantityOptions={viewQuantityOptions}
          localize={localize}
        />
      )}
      <DataDownload
        data={downloadData}
        localize={localize}
        filename={filename}
        sourceFootnoteId={sourceFootnoteId}
        sourceUrl={sourceUrl}
      />
    </div>
    <table className="text-table">
      <caption className="visually-hidden">{caption}</caption>
      <thead className="text-table-head">
        <tr className="text-table-row" key={`${parent}-header`}>
          {header.map(({ label, name, sortable }) => (
            <TextTableHeader
              ariaSort={sortColumn === name && sortState !== 'none' ? sortState : undefined}
              key={`${parent}-${name}`}
              label={label}
              localize={localize}
              name={name}
              sortable={sortable}
              sortIcon={getSortIcon(name)}
              sortIconColorName={getSortIconColor(name)}
              onSort={handleSort}
            />
          ))}
        </tr>
      </thead>
      <tbody>{tableRows}</tbody>
    </table>
  </>
```

```
  );
}

TextTable.propTypes = {
  columnToFormat: PropTypes.string,
  formatter: PropTypes.func,
  parent: PropTypes.string,
  caption: PropTypes.string,
  data: PropTypes.shape({
    header: PropTypes.arrayOf(
      PropTypes.shape({
        name: PropTypes.string,
        sortable: PropTypes.bool,
        type: PropTypes.string,
      }),
    ),
    body: PropTypes.arrayOf(PropTypes.object),
  }),
  localize: PropTypes.func,
};

TextTable.defaultProps = {
  columnToFormat: '',
  formatter: noop,
  parent: '',
  caption: '',
  data: [],
  localize: noop,
};
```

TextTableHeader - Example from Country Profiles

```
/* eslint-disable react/no-array-index-key */
/* eslint-disable react/forbid-prop-types */
/* eslint-disable react/jsx-props-no-spreading */
import PropTypes from 'prop-types';
import React, { useCallback, useMemo } from 'react';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

import { noop } from 'lodash';

export default function TextTableHeader({
  ariaSort,
  name,
  label,
  localize,
  sortable,
  sortIcon,
  sortIconColorName,
  onSort,
}) {
  const handleSort = useCallback(() => onSort(name), [name, onSort]);

  const sortIconStyle = useMemo(() => ({ color: sortIconColorName }), [sortIconColorName]);

  return (
    <th aria-sort={ariaSort} aria-label={label} scope="col">
      {sortable ? (
        <button
          aria-label={`${localize('appString', 296, 'Click to sort by')} ${label}`}
          className="text-table-sort-button"
          name={name}
          type="button"
          onClick={handleSort}
        >
          {label}
          <span aria-hidden="true">
            <FontAwesomeIcon
```

```
              className="text-table-sort-button-icon"
              icon={sortIcon}
              style={sortIconStyle}
            />
          </span>
        </button>
      ) : (
        label
      )}
    </th>
  );
}

TextTableHeader.propTypes = {
  ariaSort: PropTypes.string,
  name: PropTypes.string,
  label: PropTypes.string,
  localize: PropTypes.func,
  sortable: PropTypes.bool,
  sortIcon: PropTypes.object,
  sortIconColorName: PropTypes.string,
  onSort: PropTypes.func,
};

TextTableHeader.defaultProps = {
  ariaSort: undefined,
  name: '',
  label: '',
  localize: noop,
  sortable: false,
  sortIcon: {},
  sortIconColorName: '',
  onSort: noop,
};
```

ViewQuantityFilter - Example from Country Profiles

```
/* eslint-disable react/forbid-prop-types */
import React, { useCallback } from 'react';
import PropTypes from 'prop-types';
import { Radio } from 'antd';
import { noop } from 'lodash';

export default function ViewQuantityFilter({
  parent,
  selectedQuantity,
  setSelectedQuantity,
  viewQuantityOptions,
  localize,
}) {
  const handleChange = useCallback(
    ({ target: { value } }) => {
      setSelectedQuantity(value);
    },
    [setSelectedQuantity],
  );

  const ariaLabel = localize('appString', 295, 'Choose the number of table rows to view.');
  const text = localize('appString', 233, 'View:');

  return (
    <div className="quantity-filter-container">
      <p aria-label={ariaLabel} className="quantity-filter-label">
        {text}
      </p>
      <Radio.Group
        name={`${parent}-number-rows`}
        options={viewQuantityOptions}
        onChange={handleChange}
        value={selectedQuantity}
        optionType="button"
        buttonStyle="solid"
      />
    </div>
  );
}

ViewQuantityFilter.propTypes = {
  parent: PropTypes.string,
  selectedQuantity: PropTypes.number,
  setSelectedQuantity: PropTypes.func,
  viewQuantityOptions: PropTypes.arrayOf(PropTypes.object),
  localize: PropTypes.func,
};

ViewQuantityFilter.defaultProps = {
  parent: '',
  selectedQuantity: 20,
  setSelectedQuantity: noop,
  viewQuantityOptions: '',
  localize: noop,
};
```

DataDownload - Example from Country Profiles

```
import React, { useCallback } from 'react';
import { faDownload } from '@fortawesome/pro-light-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import Blob from 'blob';
import { saveAs } from 'file-saver';
import { noop } from 'lodash';
import PropTypes from 'prop-types';

export default function DataDownload({
  data,
  localize,
  filename,
  sourceFootnoteId,
  sourceUrl,
}) {
  const buttonText = localize('appString', 316, 'Download');

  // Replace all dash/hyphens with a new '-' because Excel does not
  // encode the dash character from the database properly.
  const footnote = localize('appString', sourceFootnoteId, '').replace(
    /[-\u2013\u2010\u2011\u2012]+/g,
    '-',
  );
  const sourceLink = sourceUrl
    ? `${localize('appString', 204, 'See related publication:')} ${sourceUrl}`
    : '';
  // Create citation as array. Add blank line at the beginning (\r\n).
  const citation = [`\r\n"${footnote}"`, `\r\n"${sourceLink}"`];

  const handleDownload = useCallback(() => {
    const csv = data
      .map((row) => row.map((cell) => (`${cell}`.includes(',') ? `"${cell}"` : cell)).join())
      .join('\n')
      .concat('\n')
      .concat(citation)
      .replace(/\r?\n/g, '\r\n');

    const blob = new Blob([csv], { type: 'text/csv;charset=utf-8' });
    saveAs(blob, filename, { autoBom: true });
  }, [data, citation, filename]);

  return (
    <button type="button" className="data-download" onClick={handleDownload}>
      <FontAwesomeIcon className="download-icon" icon={faDownload} />
      <span>{buttonText}</span>
    </button>
  );
}

DataDownload.propTypes = {
  // eslint-disable-next-line react/forbid-prop-types
  data: PropTypes.array,
  filename: PropTypes.string,
  localize: PropTypes.func,
  sourceFootnoteId: PropTypes.number,
  sourceUrl: PropTypes.string,
};

DataDownload.defaultProps = {
  data: [],
  filename: 'download.csv',
  localize: noop,
  sourceFootnoteId: null,
  sourceUrl: '',
};
```

3.Data downloads

We have two options for downloading data: via a data download button or via the download option in the vizhub template.

- Data download buttons can be added within the Text only tables as shown in point 2 above. These download buttons can also be used elsewhere in the tool.
- Our vizhub template data downloads can also be provided to screen reader users, but do need some modifications mostly around ensuring the column headers are understandable. Some tools also have additional documentation like a glossary (GBD) or definitions of the variables used in the data calculations (COVID) which would be included with the data download.

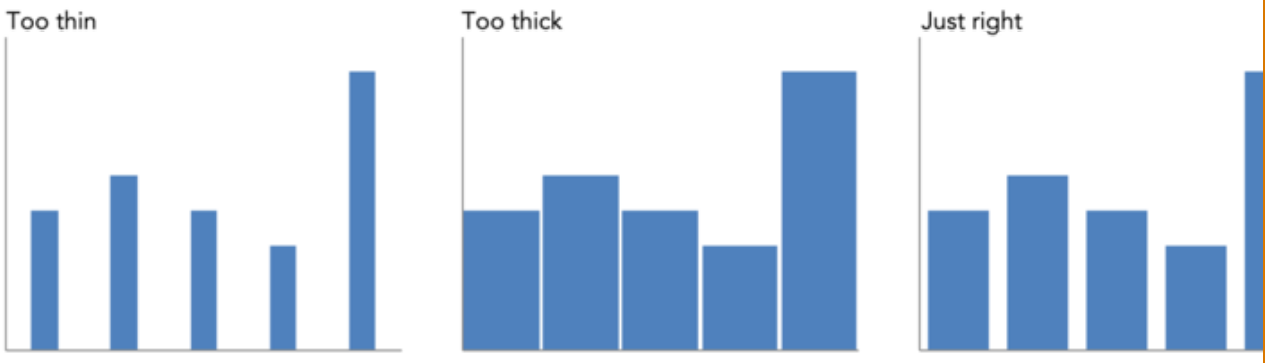
Example from WHO Rehab - Human readable headers in csv

```
// This example is from the WHO Rehabilitation tool, so it won't apply in every tool.
// Code snippets are included below for multiple files: index.php, hooks.js, util.js, TrendView.jsx

/***********/
// index.php, enable the download menu in the template
/***********/
...
$page->set('download', true);
...

/***********/
// hooks.js, define a hook for adding the csv download
/***********/
export function useCsvDownload(data, { rateOfChange = false, yearRange = [] } = {}) {
  const [metadata] = useMetadataMap();

  useEffect(() => {
        // Include human readable column headings for the csv.
    const header = [
      'Condition',
      'Location',
      'Age',
      'Sex',
      'Year',
      'Measure',
      'Mean',
      'Lower',
      'Upper',
    ];

    VizHub.config({
      download: [
        {
          text: 'Chart data (csv)',
          click() {
            const rows = data.map(
              ({
                age_id,
                condition_id,
                location_id,
                lower,
                mean,
                measure_id,
                metric_id,
                sex_id,
                upper,
                year_id,
              }) => {
                const measureLabel = rateOfChange
                  ? get(MEASURE_RATE_NAME, [measure_id, 'rateOfChange'])
                  : get(MEASURE_RATE_NAME, [measure_id, metric_id]);

                const { multiplier } = rateOfChange
                  ? get(METRIC_FORMATTER, ['rateOfChange'])
                  : get(METRIC_FORMATTER, [metric_id]);
```

```
                return [
                                  // Ensure the order of data in the rows matches the order of column
                                  // headings you defined above.
                  get(metadata, ['condition', condition_id, 'name']),
                  get(metadata, ['location', location_id, 'name']),
                  get(metadata, ['age', age_id, 'name']),
                  get(metadata, ['sex', sex_id, 'name']),
                  rateOfChange ? yearRange.join(' to ') : year_id,
                  upperFirst(measureLabel),
                  mean * multiplier,
                  lower * multiplier,
                  upper * multiplier,
                ];
              },
            );

            saveCsv([header, ...rows]);
            $('#header-actions-download').popover('closePopover');
          },
        },
      ],
    });

    return () => {
      VizHub.config({
        download: [],
      });
    };
  }, [rateOfChange, yearRange, data, metadata]);
}

/***********/
// util.js, define a utility to save the data as a csv and clean up formatting on some of the row
values.
/***********/
export function saveCsv(data, filename = 'download.csv') {
  const citation = `


"Cieza A, Causey K, Kamenov K, Hanson SW, Chatterji S, Vos T. Global estimates of the need for
rehabilitation based on the Global Burden of Disease study 2019: a systematic analysis for the Global
Burden of Disease Study 2019. The Lancet. 1 December 2020. doi: 10.1016/S0140-6736(20)32340-0.
https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(20)32340-0/fulltext."
"Available from https://vizhub.healthdata.org/rehabilitation/."
"For terms and conditions of use, please visit http://www.healthdata.org/about/terms-and-conditions."`;

  const csv = data
    .map((row) => row.map((cell) => (`${cell}`.includes(',') ? `"${cell}"` : cell)).join())
    .join('\n')
    .concat(citation)
    .replace(/\r?\n/g, '\r\n');

  const blob = new Blob([csv], { type: 'text/csv;charset=utf-8' });
  saveAs(blob, filename);
}

/***********/
// TrendView.jsx, or in any view that you want to include the csv download menu
/***********/
...
  useCsvDownload(data);
...
```

4.Appearance*

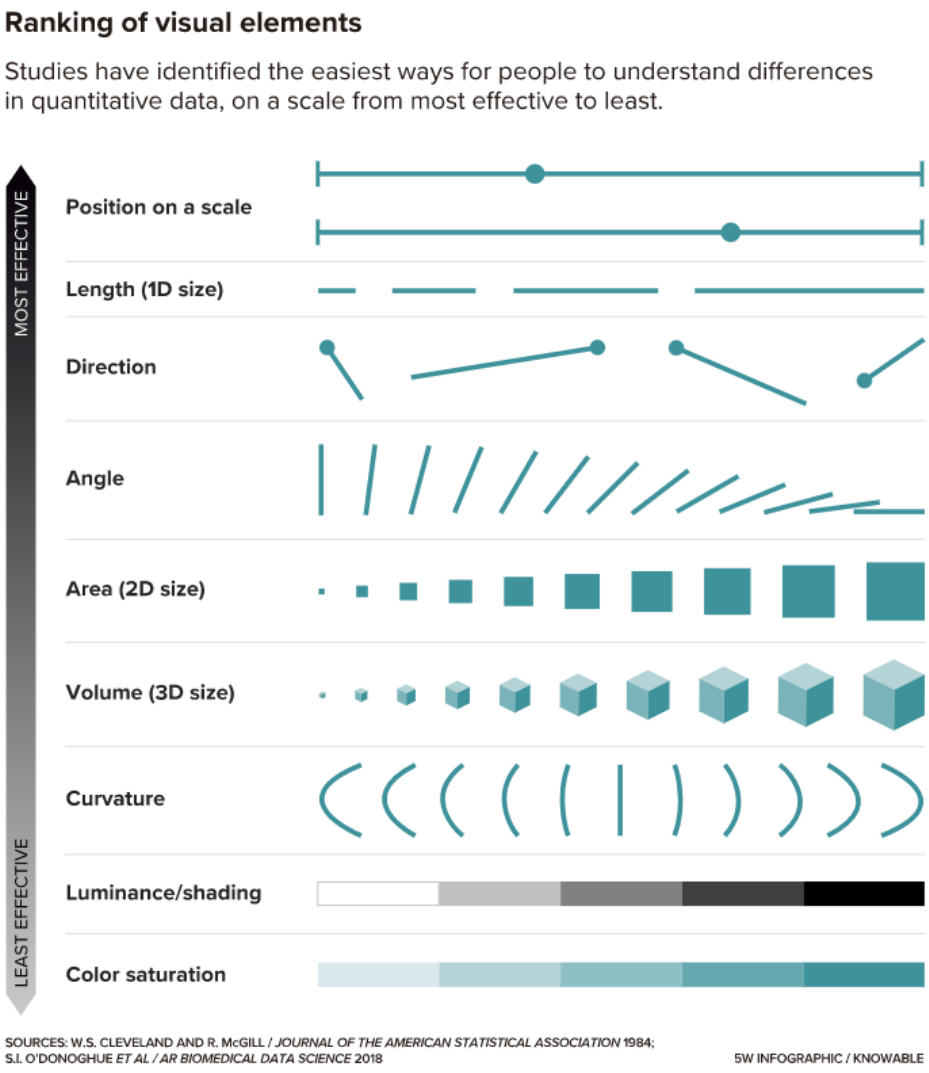
- Try to ensure chart elements and text are not overlapping.
- Add 1 px white space between different data elements (e.g., in stacked bar charts and pie charts).
- Try to prevent optical illusions by ensuring white space is used properly (not too little, not too much...you may need to experiment).



Example from Storytelling with Data by Cole Nussbaumer Knaflic

5.Use and understandability (closest is WCAG AAA 3.1.5)*

- Add explanation/guide/video to explain what the visualization is showing and how to interpret it.
    - Metadata, metrics, calculations, and variables must be defined.
    - Explain statistical confidence/uncertainty for all users including screen
- Add instructions/guide for users to understand how to interact with the visualization.
- Employ animations for object constancy. No faster than 200ms or longer than 2s.
- User imported stylesheets should not be overwritten. Perhaps we need to only add class attributes, not style attributes, via d3 and keep all style definitions in css?
- Use the right chart type that is easiest to understand. For example, see the graphic below that ranks visual elements that make it easier for people to understand quantitative differences. See the full article on how to improve scientific data visualizations.



6.Data density*

- If more than 20 elements are competing for the same space, the viz should either be split into two or more smaller, less data dense visualizations, or the data should be aggregated to a higher level with fewer elements. (This really depends on the viz and the research team, but the idea of reducing cognitive load could hopefully be used to convince the research team of the need to simplify.)
    - Maybe have supplemental data (e.g., input or outlier data points) turned off by default?
    - Maybe split the data across multiple charts so that each chart can focus on one aspect?
- If data has extremes (e.g., disparate outliers or too similar data), the viz should try to account for ways to manage these such that the data is still readable, or allow users to sort, divide, or filter the chart space. Many of our viz tools offer the ability to adjust the x and y ranges which should help mitigate these extremes.

*** Do not directly correlate to any of the WCAG 2.1 AA guidelines and are deprioritized in the effort to become WCAG AA 2.1 compliant in 2026**

Optional

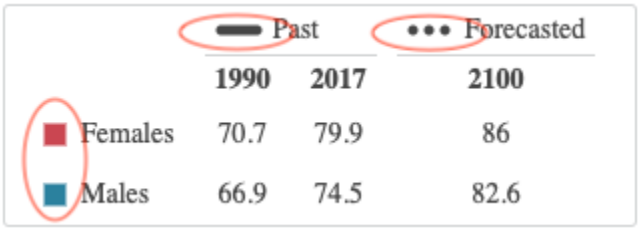
1.Making svgs accessible for screen reader users

- **If the chart includes less than 20 data points, you can try to make its content directly accessible to screen readers.**
- Adding `role=img` to the svg will trick screen readers into seeing the svg as something to read. Learn more from Sarah Fossheim's article.
    - **This has mixed results.** When adding `role=img` to the main svg in Country Profiles, the rest of the chart content became inaccessible. However, adding `role=img` to the chart data points (e.g., d3 circles with radius=0 in the line charts), **DID** help the screen readers find these elements. So, perhaps the rule is to use `role=img` on svg elements that are children of the main chart svg? Or, at least, to try it out both with and without `role=img` until you get the desired effect.
- **Axes:**
    - Add aria-labels to the axes groups and hide the tick marks and axis labels. This provides and explanation for the entire axes and prevents screen reader users from having to read over each and every tick mark.

Country Profiles - Axes aria-labels

```
// Add aria-labels to the x and y axis containers
xAxisContainer = chart.append('g')
   .attr('class', 'line-chart__axis')
   .attr('transform', `translate(${margin.left}, ${height - margin.bottom})`)
   .attr(
     'aria-label',
     `X axis showing ${axisLabels.x.text} from ${domainBounds.min} to ${domainBounds.max}.`,
   );

yAxisContainer = chart.append('g')
   .attr('class', 'line-chart__axis')
   .attr('transform', `translate(${margin.left}, ${margin.top})
   .attr(
     'aria-label',
     `Y axis showing ${axisLabels.y.text}
     from ${d3.format(formatter)(codomainBounds.min).replace('G', 'B')}
     to ${d3.format(formatter)(codomainBounds.max).replace('G', 'B')}.`,
   );

// Hide the tick marks
xAxisContainer.selectAll('.tick')
   .attr('aria-hidden', 'true');
yAxisContainer.selectAll('.tick')
   .attr('aria-hidden', 'true');

// Hide the axes labels so that the label name isn't read twice.
// (Alternately, you could let the screen reader read these axes labels
// and only include the value range in the aria-label above.)
xAxisLabel = xAxisContainer.append('text')
   .attr('class', 'line-chart__axis-label')
   .attr('text-anchor', 'middle')
   .attr('transform', `translate(${innerWidth / 2}, ${(3 / 4) * margin.bottom})`)
   .text(axisLabels.x.text)
   .attr('aria-hidden', 'true'); // Hide label from screen reader

yAxisContainer.append('text')
   .attr('class', 'line-chart__axis-label')
   .attr('text-anchor', 'middle')
   .attr(
     'transform',
     `translate(${-(3 / 4) * margin.left}, ${innerHeight / 2}) rotate(-90)`,
   )
   .text(axisLabels.y.text)
   .attr('aria-hidden', 'true');
```

- **Legends:**
    - Add an aria-label to the legend, but hide any svg element in the legend that is not helpful for a screen reader user. For example, in the Country Profile data tables/legends, the lines for Past and Forecasted and the color block for Females and Males aren't helpful for screen reader users as they are visual labels. They really only need to hear the words "Past" and "Forecasted" to understand the labels for the values.



Country Profile - Aria-labels for legends

```
function buildLegendAndTable() {
    // create table element
    const table = d3.select(`#${parentID}`).append('table')
      .attr('class', `line-table line-table--${parentID}`)
      .attr('aria-label', 'legend and data highlights'); // add aria-label for the legend
...
legendItemSvg.append('line')
      .attr(
        'class',
        d => `line-legend__shape line-legend__shape--${escapeClass(d.name)}`,
      )
      .attr('stroke', d => d.color)
      .attr('x1', `${HORIZONTAL_OFFSET}px`)
      .attr('y1', `${Math.round(SHAPE_DIMENSIONS.line.y / 2)}px`)
      .attr('x2', `${SHAPE_DIMENSIONS.line.x - HORIZONTAL_OFFSET}px`)
      .attr('y2', `${Math.round(SHAPE_DIMENSIONS.line.y / 2)}px`)
      .attr('stroke-dasharray', d => d.dashArray)
      .attr('aria-hidden', 'true'); // hide the lines for Past and Forecasted
...
const rowItemSvg = rowItemContainers.append('svg')
      .attr('width', `${SHAPE_DIMENSIONS.square.x}px`)
      .attr('height', `${SHAPE_DIMENSIONS.square.y}px`)
      .style('padding-right', '3px')
      .attr('aria-hidden', 'true'); // hide the svgs that indicate the Female/Male color
```

- **Chart content** - Add aria-labels for the chart content. In some cases (e.g., line charts or anything that uses a path instead of an element), you may need to add DOM elements to attach the aria-labels to.

Country Profiles - Line chart with accessible chart content

```
// Add invisible circles to line and attach aria-labels to them.
// This allows screen reader to read data points for EVERY year.
chartData.forEach((dataSet) => {
      const name = dataSet.name;
      lineContainer.selectAll('dot')
        .data(dataSet.data)
        .enter().append('circle')
        .attr('r', 0)
        .attr('role', 'img') // Added role=img to trick the screen reader into seeing this as a
readable element
        .attr('cx', ({ year }) => xScale(year))
        .attr('cy', ({ mean }) => yScale(mean))
        .attr(
          'aria-label',
          ({ year, mean }) =>
            `${year}, ${name}: ${util.formatWithSuffixToPrecision(1, mean)}`,
        );
});
```

- The labels will be added in the order in which the data is rendered to the chart in d3. So, you may need to reorganize or reformat the data specifically for the aria-labels. For example, the cause and risk arrow charts in Country Profiles needed to have the data reorganized so the aria-labels would read in an understandable way–a way that doesn't mimic the visual display of the chart.

Country Profiles - Data reorganization for arrow chart aria-labels

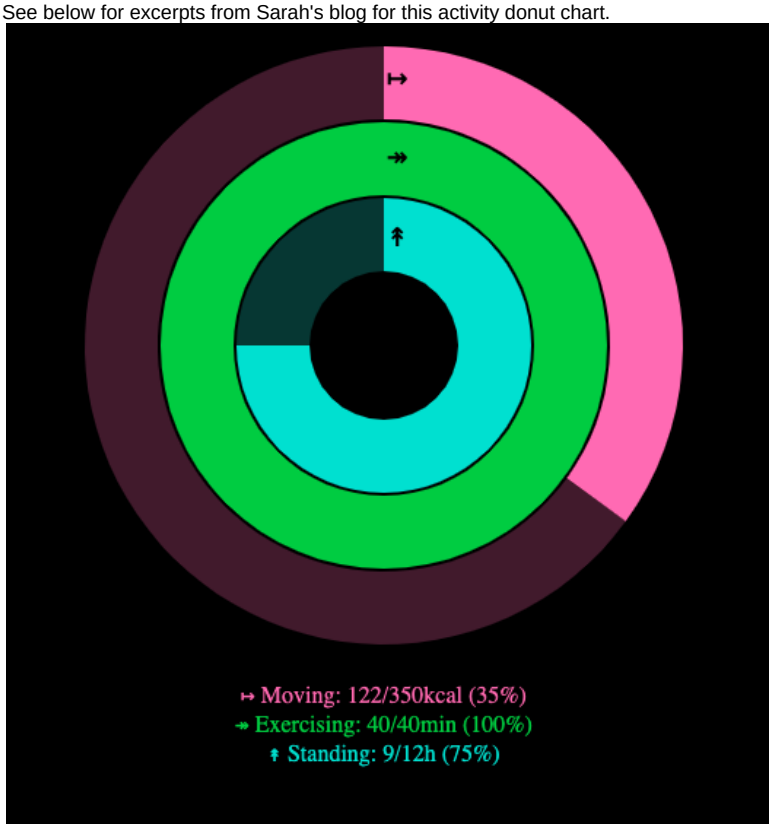
```
// Function to reorganize data for the aria-labels
function getAriaData({ end, start, percentChangeMap }) {
    return end.map(({ valueID, valueName: year2Name, type: year2Type, rank: year2Rank }) => {
      const change = `${formatter(-100 * percentChangeMap[valueID])}%`;
      return {
        year2,
        year2Name,
        year2Rank,
        year2Type,
        year1,
        year1Rank: find(start, ['valueID', valueID]).rank,
        amountAbs: change.replace('-', ''),
        changeType: (change.includes('-')) ? 'decrease' : 'increase',
      };
    });
 }

// Data generated by getAriaData is passed to the arrow chart and rendered
// on the wrapping group for each row of the arrow chart. The individual text
// elements of each row are hidden from the screen reader to reduce repetition.
function renderTableBody(data, spacers, showFirstColumn, ariaData) {
    const tableBody = svg.append('g')
      .attr('id', `${parent}-body`);

    const rowContainer = tableBody.append('g').attr('id', `${parent}-rows`);

    const rows = rowContainer.selectAll('g')
      .data(data) // this is data for the visual chart
      .enter()
      .append('g')
      .attr('id', (d, i) => `${parent}-row-${i}`);

    rowContainer.selectAll('g')
      .data(ariaData) // switch to ariaData and add aria-labels
      .attr(
        'aria-label', // add aria-labels
        ({ amountAbs, changeType, year1Rank, year2Rank, year2Type, year2Name }) => {
          let rankChange;
          if (year2Rank === year1Rank) {
            rankChange = 'did not change';
          } else {
            rankChange = year2Rank < year1Rank ? 'moved up in' : 'moved down in';
          }
          return `${year2}, rank: ${year2Rank} , ${(useRisk) ? 'Risk:' : 'Cause:'}
${year2Name}, of
          type ${TYPE_MAP[year2Type].name}, which ${rankChange} rank from a rank of
${year1Rank} in ${year1}.
          The percent change in total deaths due to ${year2Name} from ${year1} to
          ${year2} was a ${amountAbs} ${changeType}.`;
        },
      );

        if (showFirstColumn) {
      // start year value name
      rows.data(data) // switch back to visual chart data
        .append('text')
        .attr('y', (d, i) => (i + 1) * spacers.rowSpacing + (i > 9 ? spacers.otherSpacing : 0))
        .attr('x', spacers.columns[1])
        .attr('text-anchor', 'end')
        .attr('dy', '.35em')
        .text(d => d.start.valueName)
        .attr('aria-hidden', 'true'); // hide text elements that are already included in the
aria-label
        }
...
```

- Alternately, you can add aria-labels to the chart that describe the results without highlighting each individual component of the chart content. This is what was done in the FGH chart in Country Profiles. This requires that you hide the text elements of the chart content so they are not repeated out of context and out of order by the screen reader.

Country Profiles - Alternate chart content aria-labels

```
// Creates aria-labels based on the data, but organized in a way that
// hopefully makes more sense to screen reader users. Attaches these labels
// to groups that are attached to the chartContainer instead of individual chart
// elements.
function renderAriaLabels(spec) {
    chartContainer.append('g')
      .attr('class', 'fgh-chart__aria')
      .selectAll('g')
      .data(flatten(spec.stackedData))
      .enter()
      .append('g')
      .attr('aria-label',
        ({ x, dataValue, spendType }) =>
          `${x}, ${spendingTypeLookup[spendType].name}: ${spec.numberFormat(dataValue)} U.S.
dollars.`,
      );
  }

...
theLabelContainer.selectAll('g')
      .append('g')
      .attr('class', 'fgh-chart__the-label__label-text-group')
      .data(renderData)
      .enter()
      .append('text')
      .attr('class', `${textClassName}`)
      .text(({ str }) => str)
      .attr('transform', ({ d, str, lineNumber }) => labelTransformAndUpdateBounds(d,
lineNumber, str))
      .attr('aria-hidden', 'true'); // hide text labels from the screen reader
...
 d3.select(`.${selector}`)
        .append('text')
        .style('fill', ({ barColor }) => {
          if (selector.includes('_oop_')) {
            return isInsideRect ? '#503C0D' : '#957018';
          }
          return isInsideRect ? '#fff' : barColor;
        })
        .attr('class', 'fgh-chart__spend-value-labels')
        .attr('text-anchor', isInsideRect ? 'middle' : 'start')
        .text(labelText)
        .attr('transform', `translate(${trans[0]} ,${trans[1] + labelOffset})`)
        .attr('aria-hidden', 'true') // hide text labels from the screen reader
        .attr('font-family', 'AvenirNextMedium');
...
/**
   * Does the main render based on the spec, which contains sizing info
   * AND the stacked data
   * @param spec
*/
function render(spec) {
    renderSVGContainers(spec);
    renderAxes(spec);
    renderTHEBarLabels(spec); // THE = Total Health Expenditures
    renderStackedBarsWithLabels(spec);
    renderAriaLabels(spec); // fgh renders the aria-labels via this larger render function
}
```

- Adding <text> elements that describe the chart's visual elements directly to the svg will allow the screen reader to focus on and read the <text> elements.
- See below for excerpts from Sarah's blog for this activity donut chart.



Accessible svg with aria-label description

```
// function to generate a description for a donut chart
// that shows a particular activity/stat and the amount
// of time a person spends at that activity/stat.
const generateDescription = () => {
  return stats.map((stat) => {
    return `${stat.name}: ${stat.perc * 100}%.`;
  }).join(' ');
}

// Make the svg accessible and give it an aria-label
const chart = d3.select('#activityChart').append('svg')
        .attr('width', width)
        .attr('height', height)
         .attr('role', 'img') // trick screen readers
          .attr('aria-label', generateDescription()); // add aria-label to describe the chart

// Make the legend accessible with <text> elements
const legend = chart.append('text')
    .attr('text-anchor', 'middle')
    .attr('transform', `translate(${width / 2}, ${radius * 2 + 20 * (index + 2)})`)
    .attr('fill', stat.color);

  legend.append('tspan')
      .text(`${icons[stat.name.toLowerCase()]} `)
      .attr('aria-hidden', 'true'); // This hides the icons themselves from the screen reader

  legend.append('tspan')
    .text(`${stat.name}: ${stat.value}/${stat.goal}${stat.unit} (${stat.perc * 100}%)`);
```

- **Making svgs accessible (bar chart example)**
  - You can use `role=list` and `role=listitem` along with aria-labels when building the svg. Read more in Ally with Lindsay's Accessibility in D2 Bar Charts.

Accessible bar chart with aria-labels

```
// Add aria-labelledby for the chart.
const svg = d3
  .select("#chart")
  .attr('role', 'img') // Trick screen reader into reading the svg
  .attr("width", width + margin.left + margin.right + legendWidth)
  .attr("height", height + margin.top + margin.bottom)
  .attr('aria-labelledby', 'bar-chart-title'); // Pick up the title for the aria label

svg.append("text")
  .attr("x", margin.left)
  .attr("y", 250)
  .text('2018 Fruit Production')
  .attr('id', 'bar-chart-title') // Use the title in aria-labelledby

// Use role of list for the chart and add an aria-label.
const barGroups = svg
        .append('g')
        .attr('role', 'list)
        .attr('aria-label', 'bar chart')
        .attr('class', 'data')
        .attr('transform', `translate(${margin.left}, 0`);

// Add role of listitem for each bar.
const g = barGroups
        .selectAll('g')
        .data(data)
        .enter()
        .append('g')
        .attr('role', 'listitem')

// Add aria-labels to the axes.
svg
  .append("g")
  .attr("class", "x-axis")
+ .attr('aria-label', 'x axis')
  .attr("transform", `translate(${margin.left}, ${height})`)
  .call(xAxis);

svg
  .append("g")
  .attr("class", "y-axis")
+ .attr('aria-label', 'y axis')
  .attr("transform", `translate(${margin.left}, 0)`)
  .call(yAxis);
```