visualization-lecture-2-dec-batch

June 20, 2023

0.1 Data Visualization Lecture - 2

[1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import pandas as pd
import numpy as np
```

[4]:
```
data = pd.read_excel('final_vg.xlsx', sheet_name='final_vg')
```

[5]:
```
data
```

[5]:

	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales
0	2061.0	1942.0	NES	1985.0	Shooter	Capcom	4.569217	3.033887
1	9137.0	¡Shin Chan Flipa en colores!	DS	2007.0	Platform	505 Games	2.076955	1.493442
2	14279.0	.hack: Sekai no Mukou ni + Versus	PS3	2012.0	Action	Namco Bandai Games	1.145709	1.762339
3	8359.0	.hack//G.U. Vol.1//Rebirth	PS2	2006.0	Role-Playing	Namco Bandai Games	2.031986	1.389856
4	7109.0	.hack//G.U. Vol.2//Reminisce	PS2	2006.0	Role-Playing	Namco Bandai Games	2.792725	2.592054
…	…	…	…	…	…	…	…	…
16647	7925.0	Zumba Fitness Rush	X360	2012.0	Sports	505 Games	4.409308	3.167419
16648	6279.0	Zumba Fitness: World Party	Wii	2013.0	Misc	Majesco Entertainment	3.033887	2.792725
16649	6977.0	Zumba Fitness: World Party	XOne	2013.0	Misc	Majesco Entertainment	3.228043	2.004268
16650	15422.0	Zwei!!	PSP	2008.0	Role-Playing	Falcom Corporation	1.087977	0.592445
16651	12919.0	Zyuden Sentai Kyoryuger: Game de Gaburincho!!	3DS	2013.0	Action	Namco Bandai Games	1.081046	1.714664

```
       JP_Sales  Other_Sales  Global_Sales
0      3.439352     1.991671     12.802935
1      3.033887     0.394830      7.034163
2      1.493442     0.408693      4.982552
3      3.228043     0.394830      7.226880
4      1.440483     1.493442      8.363113
…         …            …            …
16647  4.168474     1.087977     13.053204
16648  1.596852     1.493442      8.878837
16649  1.833151     1.087977      7.954274
16650  1.087977     0.394830      3.509168
16651  2.004268     0.394830      5.132196

[16652 rows x 11 columns]
```

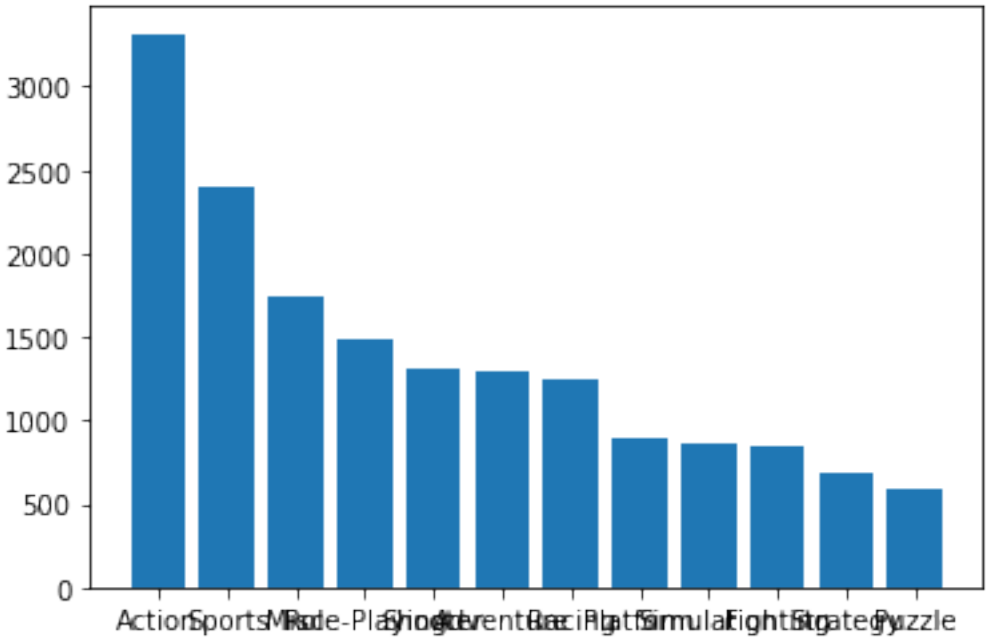
0.1.1 Univariate Analysis - Categorical data

[8]:
```
cat_counts = data['Genre'].value_counts()
cat_counts[:5]
```

[8]:
```
Action          3316
Sports          2400
Misc            1739
Role-Playing    1488
Shooter         1310
Name: Genre, dtype: int64
```

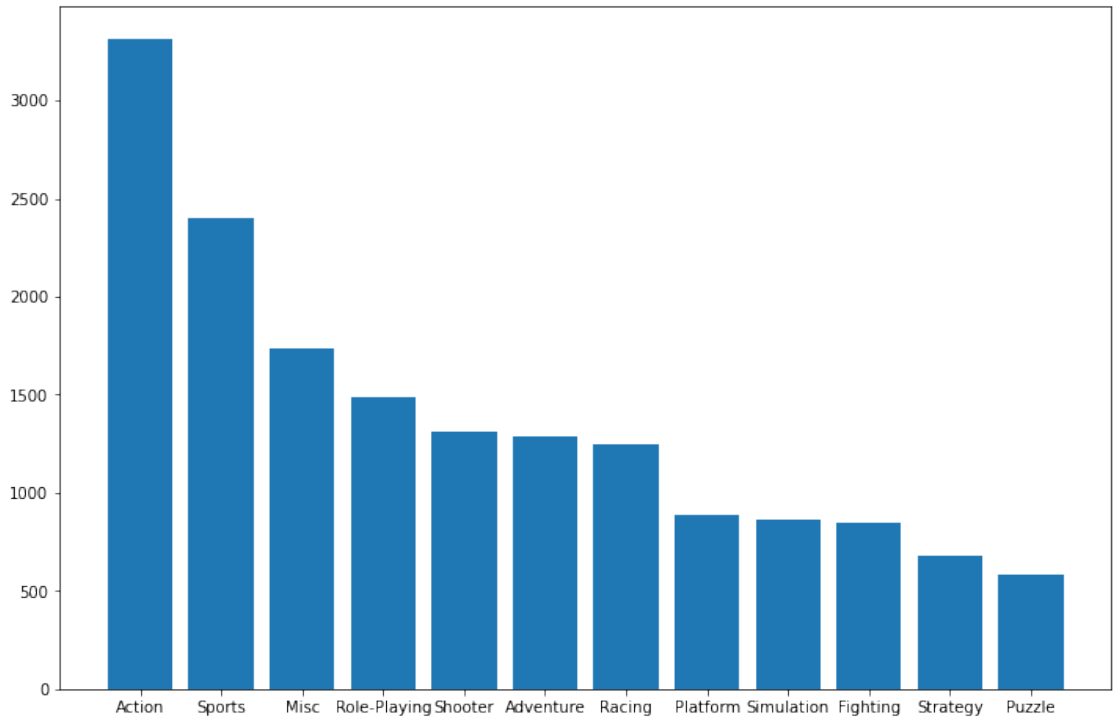
[11]:
```
x_bar = cat_counts.index
y_bar = cat_counts
plt.bar(x_bar, y_bar)
```

[11]:
```
<BarContainer object of 12 artists>
```

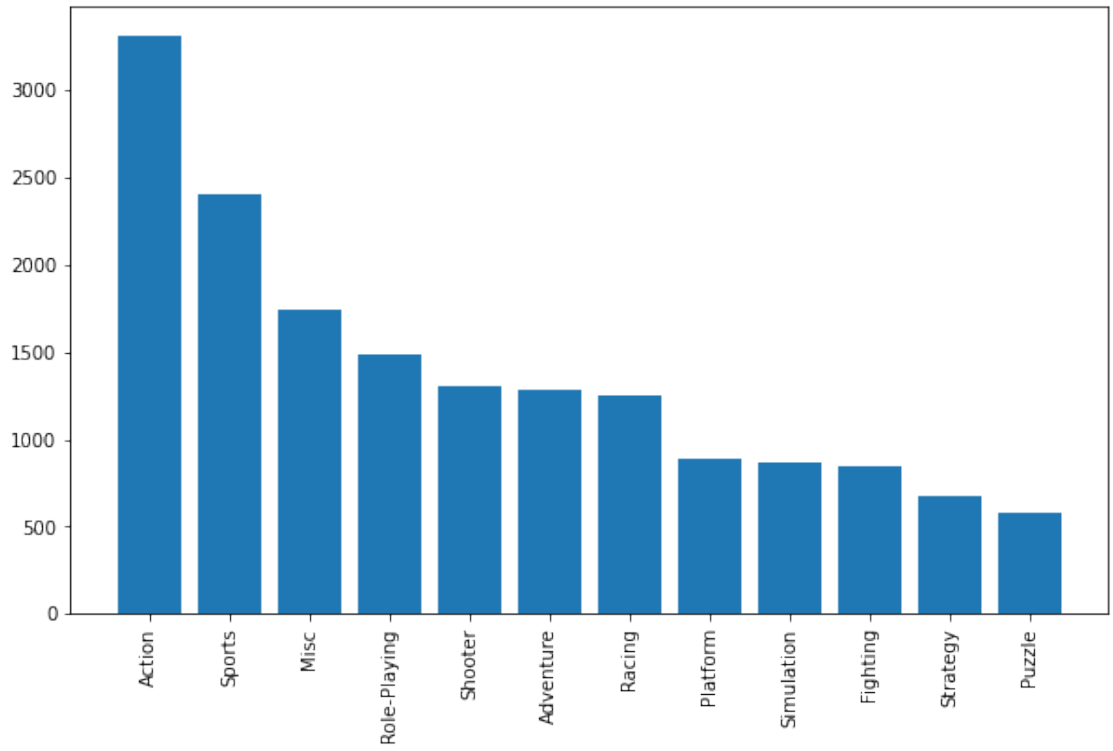


[14]:
```
plt.figure(figsize=(12, 8))
plt.bar(x_bar, y_bar)
```

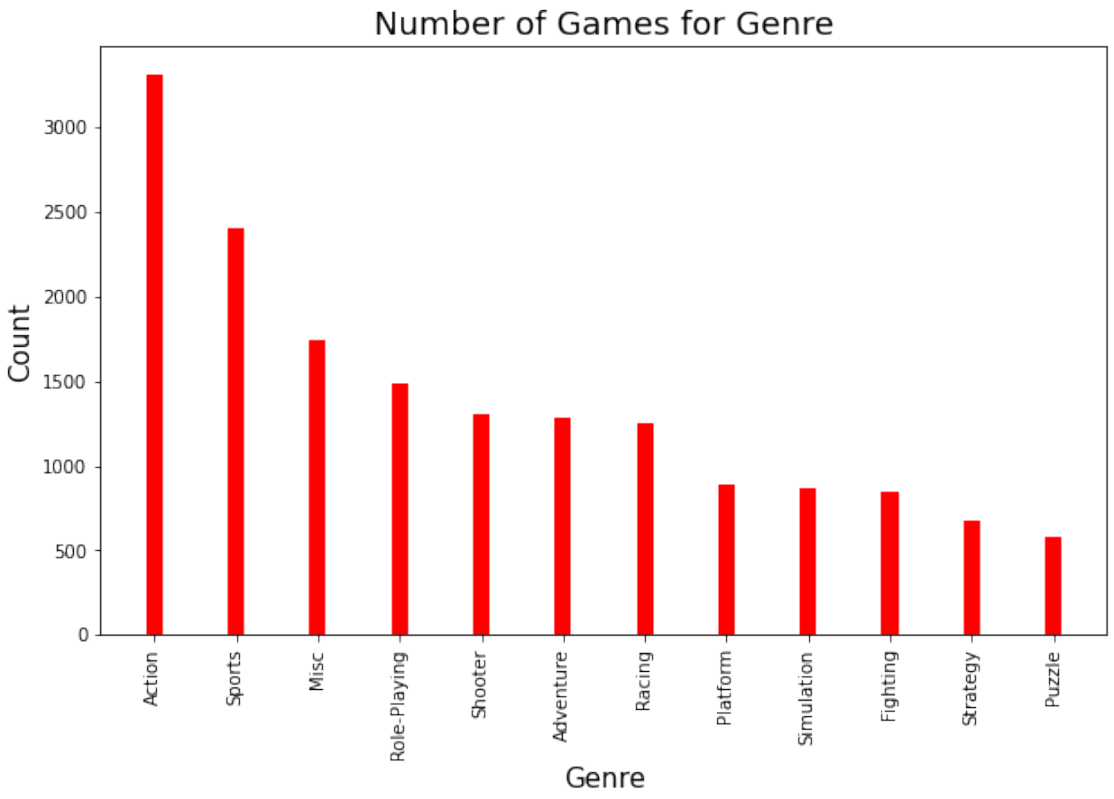
[14]: <BarContainer object of 12 artists>



[18]:
```
plt.figure(figsize=(10, 6))
plt.bar(x_bar, y_bar)
plt.xticks(rotation=90, fontsize=10)
plt.show()
```



[25]:
```
plt.figure(figsize=(10, 6))
plt.bar(x_bar, y_bar, width=0.2, color='red')
plt.xticks(rotation=90, fontsize=10)
plt.xlabel('Genre', fontsize=15)
plt.ylabel('Count', fontsize=15)
plt.title('Number of Games for Genre', fontsize=18)
plt.show()
```



[26]:
```
help(plt.bar)
```

```
Help on function bar in module matplotlib.pyplot:

bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)
    Make a bar plot.

    The bars are positioned at *x* with the given *align*\ment. Their
    dimensions are given by *height* and *width*. The vertical baseline
    is *bottom* (default 0).

    Many parameters can take either a single value applying to all bars
    or a sequence of values, one for each bar.

    Parameters
    ----------
    x : float or array-like
        The x coordinates of the bars. See also *align* for the
        alignment of the bars to the coordinates.

    height : float or array-like
        The height(s) of the bars.
```

```
width : float or array-like, default: 0.8
    The width(s) of the bars.

bottom : float or array-like, default: 0
    The y coordinate(s) of the bars bases.

align : {'center', 'edge'}, default: 'center'
    Alignment of the bars to the *x* coordinates:

    - 'center': Center the base on the *x* positions.
    - 'edge': Align the left edges of the bars with the *x* positions.

    To align the bars on the right edge pass a negative *width* and
    ``align='edge'``.

Returns
-------
`.BarContainer`
    Container with all the bars and optionally errorbars.

Other Parameters
----------------
color : color or list of color, optional
    The colors of the bar faces.

edgecolor : color or list of color, optional
    The colors of the bar edges.

linewidth : float or array-like, optional
    Width of the bar edge(s). If 0, don't draw edges.

tick_label : str or list of str, optional
    The tick labels of the bars.
    Default: None (Use default numeric labels.)

xerr, yerr : float or array-like of shape(N,) or shape(2, N), optional
    If not *None*, add horizontal / vertical errorbars to the bar tips.
    The values are +/- sizes relative to the data:

    - scalar: symmetric +/- values for all bars
    - shape(N,): symmetric +/- values for each bar
    - shape(2, N): Separate - and + values for each bar. First row
      contains the lower errors, the second row contains the upper
      errors.
    - *None*: No errorbar. (Default)

    See :doc:`/gallery/statistics/errorbar_features`
```

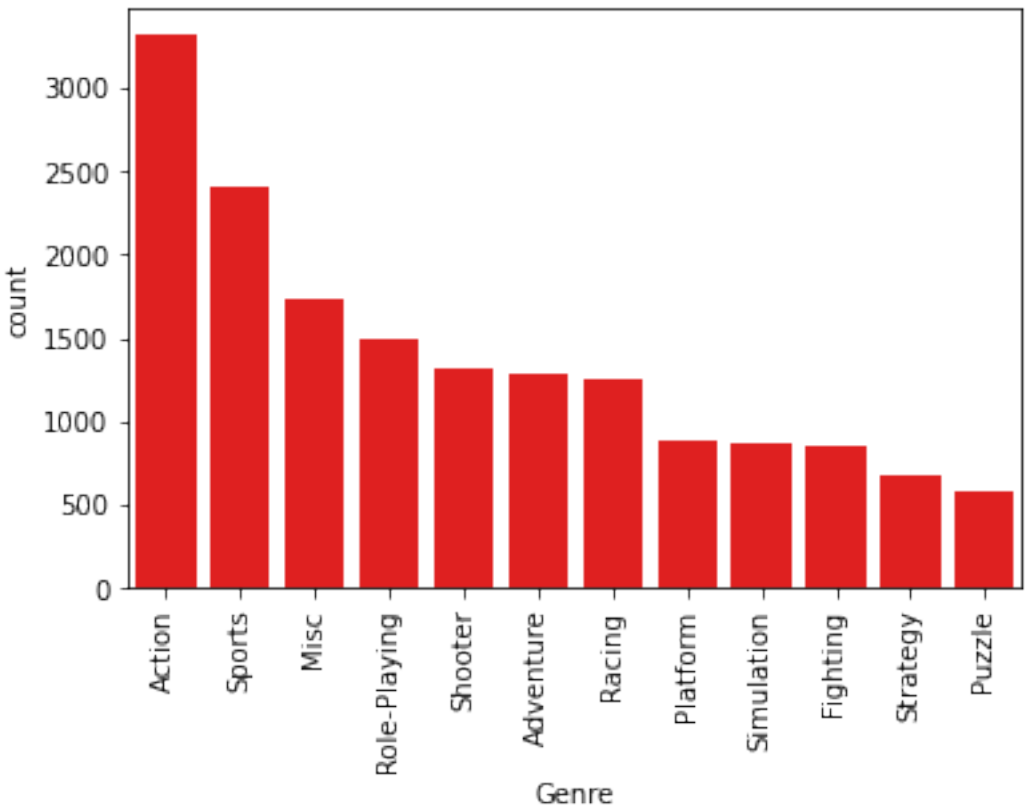
```
        for an example on the usage of ``xerr`` and ``yerr``.

    ecolor : color or list of color, default: 'black'
        The line color of the errorbars.

    capsize : float, default: :rc:`errorbar.capsize`
       The length of the error bar caps in points.

    error_kw : dict, optional
        Dictionary of kwargs to be passed to the `~.Axes.errorbar`
        method. Values of *ecolor* or *capsize* defined here take
        precedence over the independent kwargs.

    log : bool, default: False
        If *True*, set the y-axis to be log scale.

    data : indexable object, optional
        If given, all parameters also accept a string ``s``, which is
        interpreted as ``data[s]`` (unless this raises an exception).

    **kwargs : `.Rectangle` properties

    Properties:
        agg_filter: a filter function, which takes a (m, n, 3) float array and a
dpi value, and returns a (m, n, 3) array
        alpha: scalar or None
        angle: unknown
        animated: bool
        antialiased or aa: bool or None
        bounds: (left, bottom, width, height)
        capstyle: `.CapStyle` or {'butt', 'projecting', 'round'}
        clip_box: `.Bbox`
        clip_on: bool
        clip_path: Patch or (Path, Transform) or None
        color: color
        edgecolor or ec: color or None
        facecolor or fc: color or None
        figure: `.Figure`
        fill: bool
        gid: str
        hatch: {'/', '\\', '|', '-', '+', 'x', 'o', 'O', '.', '*'}
        height: unknown
        in_layout: bool
        joinstyle: `.JoinStyle` or {'miter', 'round', 'bevel'}
        label: object
        linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), …}
        linewidth or lw: float or None
        path_effects: `.AbstractPathEffect`
```

```
        picker: None or bool or float or callable
        rasterized: bool
        sketch_params: (scale: float, length: float, randomness: float)
        snap: bool or None
        transform: `.Transform`
        url: str
        visible: bool
        width: unknown
        x: unknown
        xy: (float, float)
        y: unknown
        zorder: float

    See Also
    --------
    barh : Plot a horizontal bar plot.

    Notes
    -----
    Stacked bars can be achieved by passing individual *bottom* values per
    bar. See :doc:`/gallery/lines_bars_and_markers/bar_stacked`.
```

[30]:
```
cat_counts.index
```

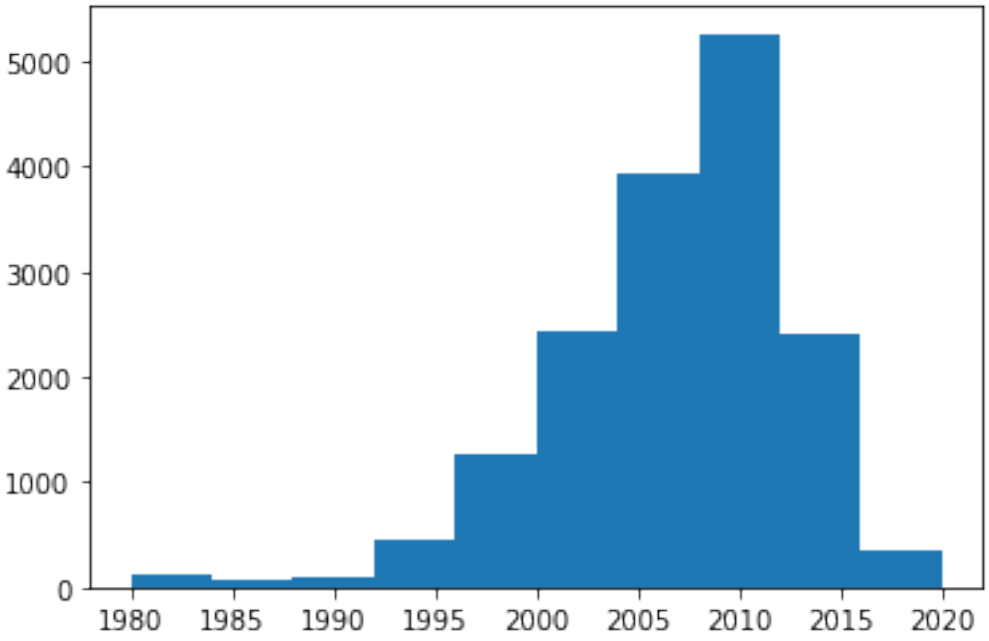
[30]:
```
Index(['Action', 'Sports', 'Misc', 'Role-Playing', 'Shooter', 'Adventure',
       'Racing', 'Platform', 'Simulation', 'Fighting', 'Strategy', 'Puzzle'],
      dtype='object')
```

[32]:
```
sns.countplot(data=data, x='Genre', order=cat_counts.index, color='red')
plt.xticks(rotation=90)
plt.show()
```

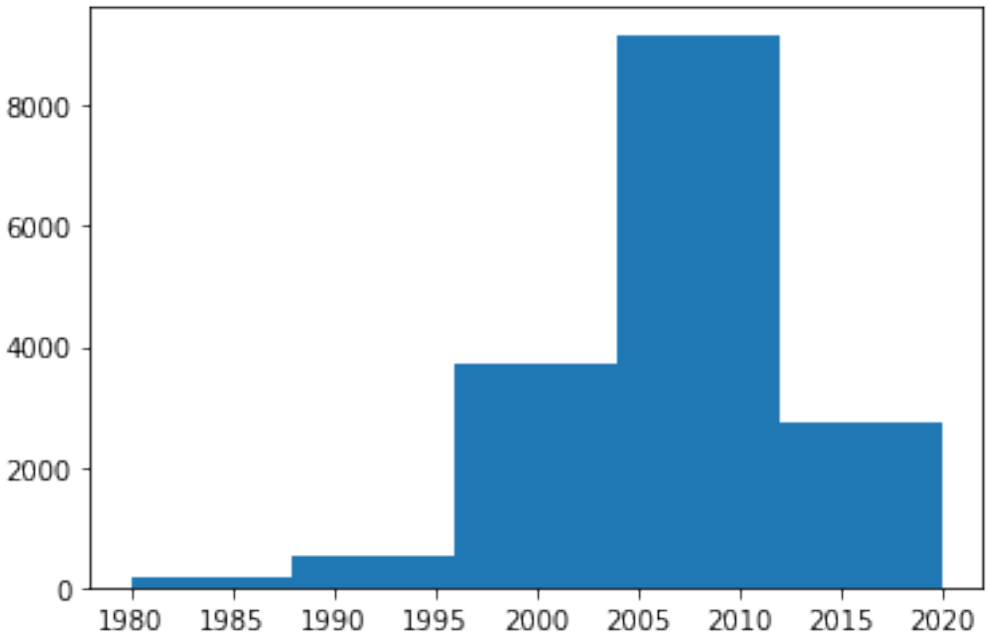


0.1.2 Univariate Analysis- Numerical Data

[33]:
```
plt.hist(data['Year'])
plt.show()
```

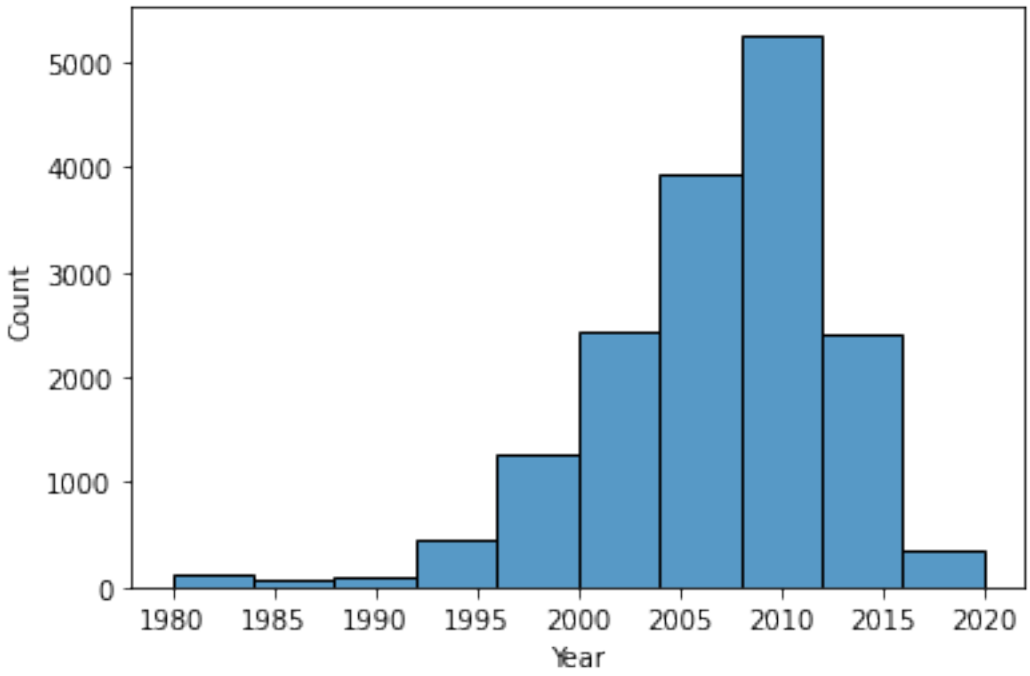


[34]:
```
plt.hist(data['Year'], bins=5)
plt.show()
```



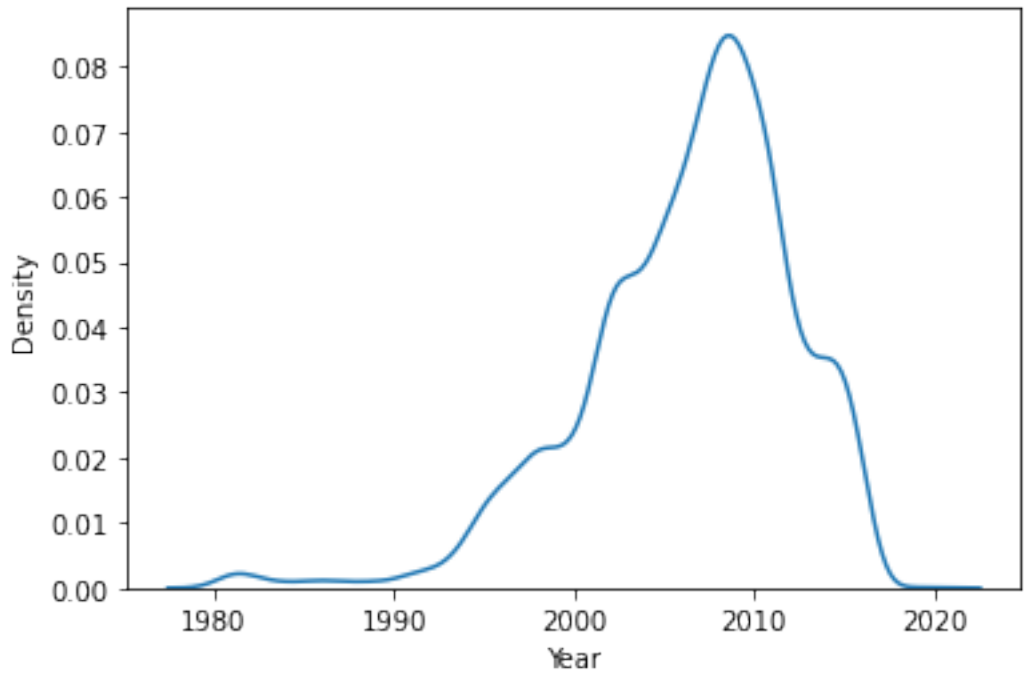
[35]:
```
sns.histplot(data['Year'], bins=10)
```

[35]: <AxesSubplot:xlabel='Year', ylabel='Count'>



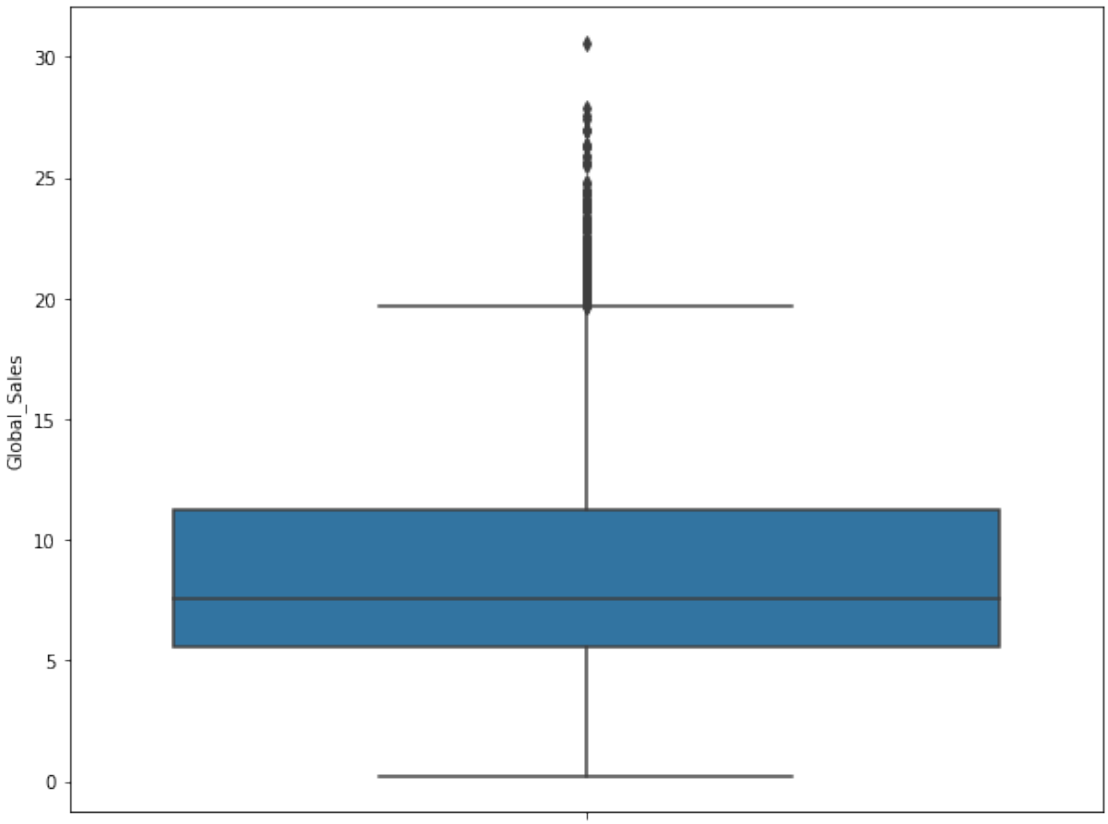
[36]:
```
sns.kdeplot(data['Year'])
```

[36]: <AxesSubplot:xlabel='Year', ylabel='Density'>



[38]:
```
plt.figure(figsize=(10, 8))
sns.boxplot(y=data['Global_Sales'])
```

[38]:
```
<AxesSubplot:ylabel='Global_Sales'>
```



0.2 Bi Variate Analysis - (Numerical - Numerical)

[39]:
```
data['Name'].value_counts()
```

[39]:
```
Ice Hockey                          41
Baseball                            17
Need for Speed: Most Wanted         12
Ratatouille                          9
FIFA 14                              9
                                    ..
Indy 500                             1
Indy Racing 2000                     1
Indycar Series 2005                  1
```

```
inFAMOUS                                          1
Zyuden Sentai Kyoryuger: Game de Gaburincho!!     1
Name: Name, Length: 11493, dtype: int64
```

[43]:
```
ih
```

[43]:

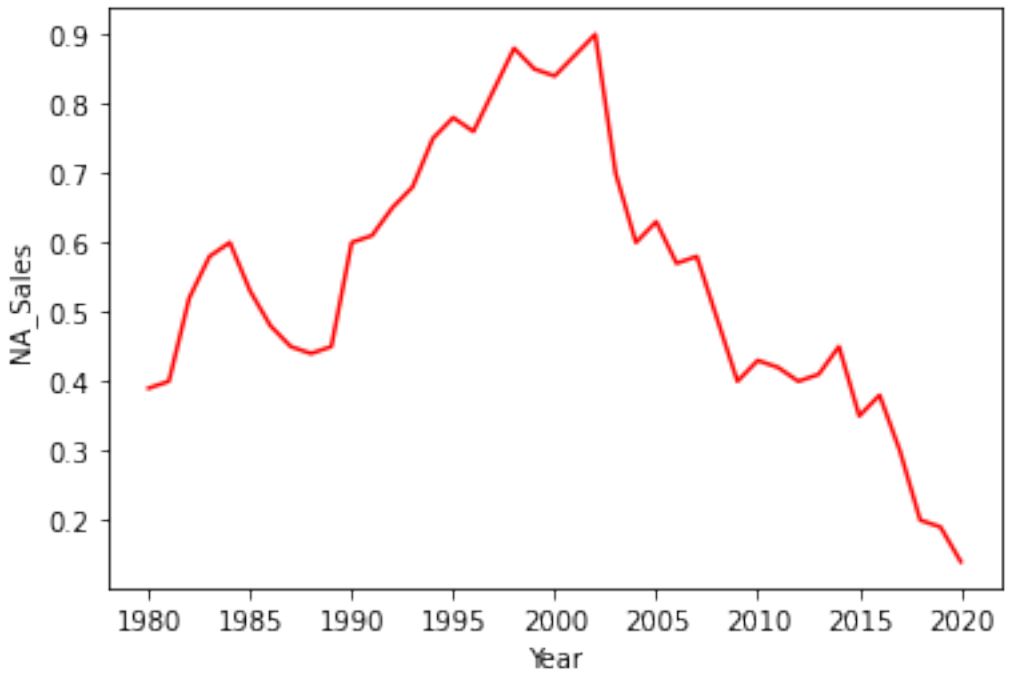
	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	\
6073	639.0	Ice Hockey	NES	1988.0	Sports	Nintendo	0.44	
6074	4027.0	Ice Hockey	2600.0	1980.0	Sports	Activision	0.39	
6075	4149.0	Ice Hockey	2600.0	1991.0	Sports	Activision	0.61	
6076	4149.0	Ice Hockey	2600.0	1992.0	Sports	Activision	0.65	
6077	4149.0	Ice Hockey	SNES	1993.0	Sports	Activision	0.68	
6078	4149.0	Ice Hockey	SNES	1994.0	Sports	Activision	0.75	
6079	4149.0	Ice Hockey	SNES	1995.0	Sports	Activision	0.78	
6080	4149.0	Ice Hockey	SNES	1996.0	Sports	Activision	0.76	
6081	4149.0	Ice Hockey	SNES	1997.0	Sports	Activision	0.82	
6082	4149.0	Ice Hockey	SNES	1998.0	Sports	Activision	0.88	
6083	4149.0	Ice Hockey	SNES	1999.0	Sports	Activision	0.85	
6084	4148.0	Ice Hockey	SNES	2000.0	Sports	Activision	0.84	
6085	3000.0	Ice Hockey	SNES	2001.0	Sports	Activision	0.87	
6086	3012.0	Ice Hockey	SNES	2002.0	Sports	Activision	0.90	
6087	3013.0	Ice Hockey	SNES	2003.0	Sports	Activision	0.70	
6088	3013.0	Ice Hockey	SNES	2004.0	Sports	Activision	0.60	
6089	3011.0	Ice Hockey	PC	2005.0	Sports	Activision	0.63	
6090	1234.0	Ice Hockey	PC	2006.0	Sports	Activision	0.57	
6091	3012.0	Ice Hockey	PC	2007.0	Sports	Activision	0.58	
6092	1231.0	Ice Hockey	PC	2008.0	Sports	Activision	0.49	
6093	3012.0	Ice Hockey	PC	2009.0	Sports	Activision	0.40	
6094	3012.0	Ice Hockey	PC	2010.0	Sports	Activision	0.43	
6095	3012.0	Ice Hockey	PC	2011.0	Sports	Activision	0.42	
6096	3012.0	Ice Hockey	PC	2012.0	Sports	Activision	0.40	
6097	3012.0	Ice Hockey	PC	2013.0	Sports	Activision	0.41	
6098	3012.0	Ice Hockey	PS	2014.0	Sports	Activision	0.45	
6099	3012.0	Ice Hockey	PS	2015.0	Sports	Activision	0.35	
6100	3012.0	Ice Hockey	PS	2016.0	Sports	Activision	0.38	
6101	3012.0	Ice Hockey	PS	2017.0	Sports	Activision	0.30	
6102	3012.0	Ice Hockey	PS	2018.0	Sports	Activision	0.20	
6103	3012.0	Ice Hockey	PS	2019.0	Sports	Activision	0.19	
6104	3012.0	Ice Hockey	PS	2020.0	Sports	Activision	0.14	
6105	4100.0	Ice Hockey	2600.0	1990.0	Sports	Activision	0.60	
6106	4519.0	Ice Hockey	2600.0	1981.0	Sports	Activision	0.40	
6107	3645.0	Ice Hockey	2600.0	1982.0	Sports	Activision	0.52	
6108	5384.0	Ice Hockey	2600.0	1983.0	Sports	Activision	0.58	
6109	6299.0	Ice Hockey	2600.0	1984.0	Sports	Activision	0.60	
6110	5084.0	Ice Hockey	2600.0	1986.0	Sports	Activision	0.48	
6111	6941.0	Ice Hockey	2600.0	1987.0	Sports	Activision	0.45	
6112	4149.0	Ice Hockey	2600.0	1989.0	Sports	Activision	0.45	

```
6113  4140.0  Ice Hockey   2600.0  1985.0  Sports  Activision      0.53
```

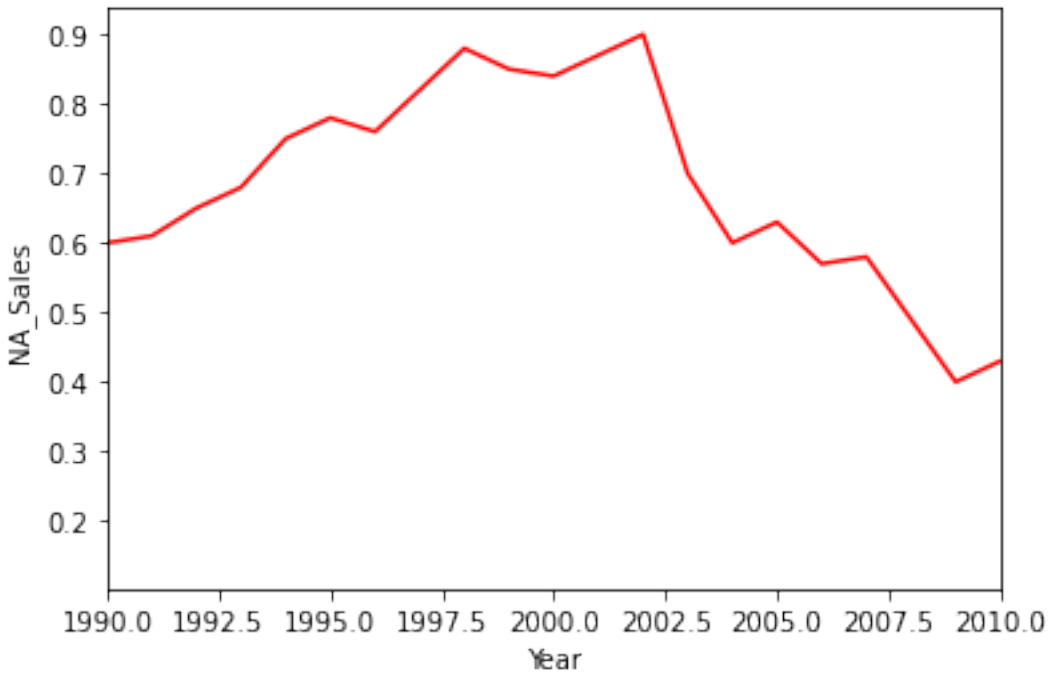
	EU_Sales	JP_Sales	Other_Sales	Global_Sales
6073	3.860566	4.751539	2.004268	15.855389
6074	1.493442	2.741701	0.394830	4.956249
6075	0.020000	0.000000	0.010000	0.470000
6076	0.020000	0.000000	0.010000	0.470000
6077	0.020000	0.000000	0.010000	0.470000
6078	0.020000	0.000000	0.010000	0.470000
6079	0.020000	0.000000	0.010000	0.470000
6080	0.020000	0.000000	0.010000	0.470000
6081	0.020000	0.000000	0.010000	0.470000
6082	0.020000	0.000000	0.010000	0.470000
6083	0.020000	0.000000	0.010000	0.470000
6084	0.020000	0.000000	0.010000	0.470000
6085	0.020000	0.000000	0.010000	0.470000
6086	0.020000	0.000000	0.010000	0.470000
6087	0.020000	0.000000	0.010000	0.470000
6088	0.020000	0.000000	0.010000	0.470000
6089	0.020000	0.000000	0.010000	0.470000
6090	0.020000	0.000000	0.010000	0.470000
6091	0.020000	0.000000	0.010000	0.470000
6092	0.020000	0.000000	0.010000	0.470000
6093	0.020000	0.000000	0.010000	0.470000
6094	0.020000	0.000000	0.010000	0.470000
6095	0.020000	0.000000	0.010000	0.470000
6096	0.020000	0.000000	0.010000	0.470000
6097	0.020000	0.000000	0.010000	0.470000
6098	0.020000	0.000000	0.010000	0.470000
6099	0.020000	0.000000	0.010000	0.470000
6100	0.020000	0.000000	0.010000	0.470000
6101	0.020000	0.000000	0.010000	0.470000
6102	0.020000	0.000000	0.010000	0.470000
6103	0.020000	0.000000	0.010000	0.470000
6104	0.020000	0.000000	0.010000	0.470000
6105	0.020000	0.000000	0.010000	0.470000
6106	0.020000	0.000000	0.000000	0.430000
6107	0.030000	0.000000	0.010000	0.550000
6108	0.020000	0.000000	0.000000	0.340000
6109	0.010000	0.000000	0.000000	0.270000
6110	0.020000	0.000000	0.000000	0.370000
6111	0.010000	0.000000	0.000000	0.240000
6112	0.020000	0.000000	0.010000	0.470000
6113	0.020000	0.000000	0.010000	0.470000

[46]:
```
ih = data.loc[data['Name']=='Ice Hockey']
sns.lineplot(data=ih, x='Year', y='NA_Sales', color='red')
```

```
plt.show()
```



[47]:
```
ih = data.loc[data['Name']=='Ice Hockey']
sns.lineplot(data=ih, x='Year', y='NA_Sales', color='red')
plt.xlim(left=1990, right=2010)
plt.show()
```



[48]:
```
sns.lineplot?
```

```
Signature:
sns.lineplot(
    *,
    x=None,
    y=None,
    hue=None,
    size=None,
    style=None,
    data=None,
    palette=None,
    hue_order=None,
    hue_norm=None,
    sizes=None,
    size_order=None,
    size_norm=None,
    dashes=True,
    markers=None,
    style_order=None,
    units=None,
    estimator='mean',
    ci=95,
    n_boot=1000,
    seed=None,
```

```
    sort=True,
    err_style='band',
    err_kws=None,
    legend='auto',
    ax=None,
    **kwargs,
)
Docstring:
Draw a line plot with possibility of several semantic groupings.

The relationship between ``x`` and ``y`` can be shown for different subsets
of the data using the ``hue``, ``size``, and ``style`` parameters. These
parameters control what visual semantics are used to identify the different
subsets. It is possible to show up to three dimensions independently by
using all three semantic types, but this style of plot can be hard to
interpret and is often ineffective. Using redundant semantics (i.e. both
``hue`` and ``style`` for the same variable) can be helpful for making
graphics more accessible.

See the :ref:`tutorial <relational_tutorial>` for more information.

The default treatment of the ``hue`` (and to a lesser extent, ``size``)
semantic, if present, depends on whether the variable is inferred to
represent "numeric" or "categorical" data. In particular, numeric variables
are represented with a sequential colormap by default, and the legend
entries show regular "ticks" with values that may or may not exist in the
data. This behavior can be controlled through various parameters, as
described and illustrated below.

By default, the plot aggregates over multiple ``y`` values at each value of
``x`` and shows an estimate of the central tendency and a confidence
interval for that estimate.

Parameters
----------
x, y : vectors or keys in ``data``
    Variables that specify positions on the x and y axes.
hue : vector or key in ``data``
    Grouping variable that will produce lines with different colors.
    Can be either categorical or numeric, although color mapping will
    behave differently in latter case.
size : vector or key in ``data``
    Grouping variable that will produce lines with different widths.
    Can be either categorical or numeric, although size mapping will
    behave differently in latter case.
style : vector or key in ``data``
    Grouping variable that will produce lines with different dashes
    and/or markers. Can have a numeric dtype but will always be treated
```

```
    as categorical.
data : :class:`pandas.DataFrame`, :class:`numpy.ndarray`, mapping, or sequence
    Input data structure. Either a long-form collection of vectors that can be
    assigned to named variables or a wide-form dataset that will be internally
    reshaped.
palette : string, list, dict, or :class:`matplotlib.colors.Colormap`
    Method for choosing the colors to use when mapping the ``hue`` semantic.
    String values are passed to :func:`color_palette`. List or dict values
    imply categorical mapping, while a colormap object implies numeric mapping.
hue_order : vector of strings
    Specify the order of processing and plotting for categorical levels of the
    ``hue`` semantic.
hue_norm : tuple or :class:`matplotlib.colors.Normalize`
    Either a pair of values that set the normalization range in data units
    or an object that will map from data units into a [0, 1] interval. Usage
    implies numeric mapping.
sizes : list, dict, or tuple
    An object that determines how sizes are chosen when ``size`` is used.
    It can always be a list of size values or a dict mapping levels of the
    ``size`` variable to sizes. When ``size``  is numeric, it can also be
    a tuple specifying the minimum and maximum size to use such that other
    values are normalized within this range.
size_order : list
    Specified order for appearance of the ``size`` variable levels,
    otherwise they are determined from the data. Not relevant when the
    ``size`` variable is numeric.
size_norm : tuple or Normalize object
    Normalization in data units for scaling plot objects when the
    ``size`` variable is numeric.
dashes : boolean, list, or dictionary
    Object determining how to draw the lines for different levels of the
    ``style`` variable. Setting to ``True`` will use default dash codes, or
    you can pass a list of dash codes or a dictionary mapping levels of the
    ``style`` variable to dash codes. Setting to ``False`` will use solid
    lines for all subsets. Dashes are specified as in matplotlib: a tuple
    of ``(segment, gap)`` lengths, or an empty string to draw a solid line.
markers : boolean, list, or dictionary
    Object determining how to draw the markers for different levels of the
    ``style`` variable. Setting to ``True`` will use default markers, or
    you can pass a list of markers or a dictionary mapping levels of the
    ``style`` variable to markers. Setting to ``False`` will draw
    marker-less lines.  Markers are specified as in matplotlib.
style_order : list
    Specified order for appearance of the ``style`` variable levels
    otherwise they are determined from the data. Not relevant when the
    ``style`` variable is numeric.
units : vector or key in ``data``
    Grouping variable identifying sampling units. When used, a separate
```

```
    line will be drawn for each unit with appropriate semantics, but no
    legend entry will be added. Useful for showing distribution of
    experimental replicates when exact identities are not needed.
estimator : name of pandas method or callable or None
    Method for aggregating across multiple observations of the ``y``
    variable at the same ``x`` level. If ``None``, all observations will
    be drawn.
ci : int or "sd" or None
    Size of the confidence interval to draw when aggregating with an
    estimator. "sd" means to draw the standard deviation of the data.
    Setting to ``None`` will skip bootstrapping.
n_boot : int
    Number of bootstraps to use for computing the confidence interval.
seed : int, numpy.random.Generator, or numpy.random.RandomState
    Seed or random number generator for reproducible bootstrapping.
sort : boolean
    If True, the data will be sorted by the x and y variables, otherwise
    lines will connect points in the order they appear in the dataset.
err_style : "band" or "bars"
    Whether to draw the confidence intervals with translucent error bands
    or discrete error bars.
err_kws : dict of keyword arguments
    Additional paramters to control the aesthetics of the error bars. The
    kwargs are passed either to :meth:`matplotlib.axes.Axes.fill_between`
    or :meth:`matplotlib.axes.Axes.errorbar`, depending on ``err_style``.
legend : "auto", "brief", "full", or False
    How to draw the legend. If "brief", numeric ``hue`` and ``size``
    variables will be represented with a sample of evenly spaced values.
    If "full", every group will get an entry in the legend. If "auto",
    choose between brief or full representation based on number of levels.
    If ``False``, no legend data is added and no legend is drawn.
ax : :class:`matplotlib.axes.Axes`
    Pre-existing axes for the plot. Otherwise, call :func:`matplotlib.pyplot.gca`
    internally.
kwargs : key, value mappings
    Other keyword arguments are passed down to
    :meth:`matplotlib.axes.Axes.plot`.

Returns
-------
:class:`matplotlib.axes.Axes`
    The matplotlib axes containing the plot.

See Also
--------
scatterplot : Plot data using points.
pointplot : Plot point estimates and CIs using markers and lines.
```

```
Examples
--------

.. include:: ../docstrings/lineplot.rst
File:      /usr/local/lib/python3.9/site-packages/seaborn/relational.py
Type:      function
```

[52]:
```
ih = data.loc[data['Name']=='Ice Hockey']
baseball = data.loc[data['Name']=='Baseball']
sns.lineplot(data=ih, x='Year', y='NA_Sales', color='red' )
sns.lineplot(data=baseball, x='Year', y='NA_Sales' )
plt.legend(['Ice hockey', 'Base ball'])
plt.show()
```