data vizualization with pandas :

key features:

1.variety o f plot type: line plot , bar plot , histogram, box plot , and scatter plots.

2. customize :user can customize plots by adding titles , labels and styling ..for visualization

3.Handling of Missing Data: Pandas efficiently handles missing data ensuring that visualizations accurately represent the dataset without errors.

4.Integration with Matplotlib: Pandas integrates with Matplotlib that allow users to create a wide range of static, animated, and interactive plots.

In [3]:
```
import numpy as np
import pandas as pd

df1 = pd.read_csv('df1', index_col=0)
df2 = pd.read_csv('df2')
```

1. Line Plots using Pandas DataFrame

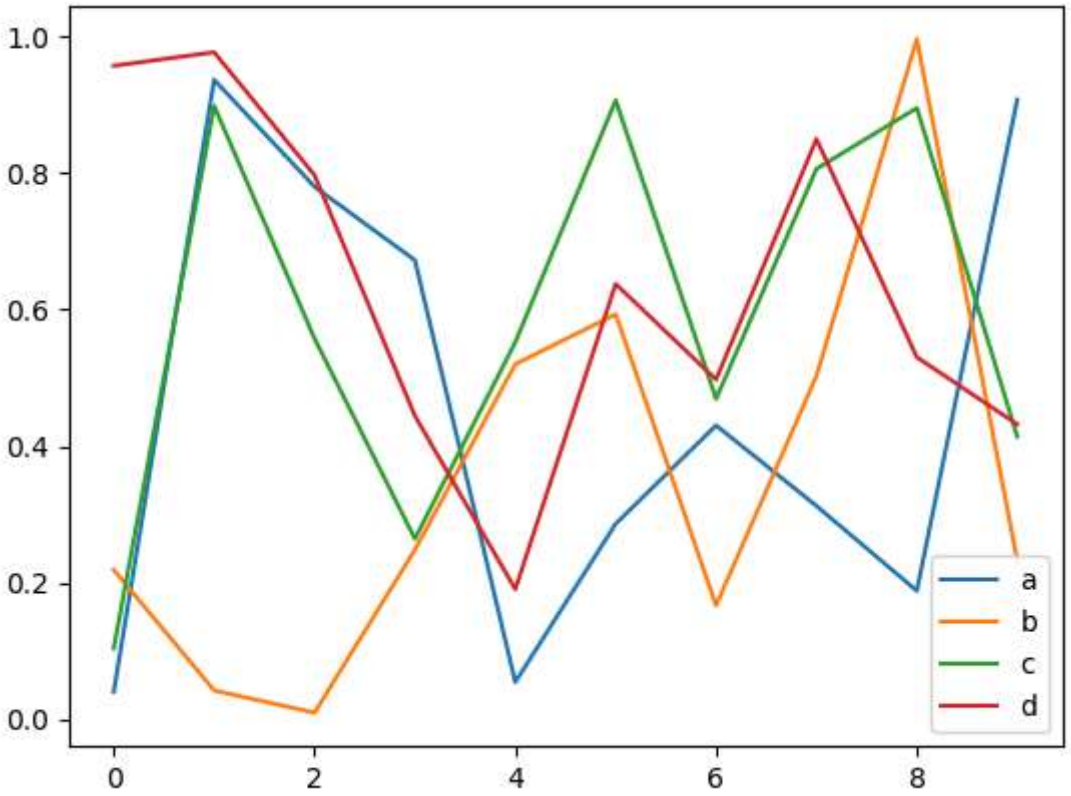
A Line plot is a graph that shows the frequency of data along a number line.

It is best to use a line plot when the data is time series.

It can be created using Dataframe.plot() function.

In [5]:
```
df2.plot()
```

Out[5]:
```
<Axes: >
```



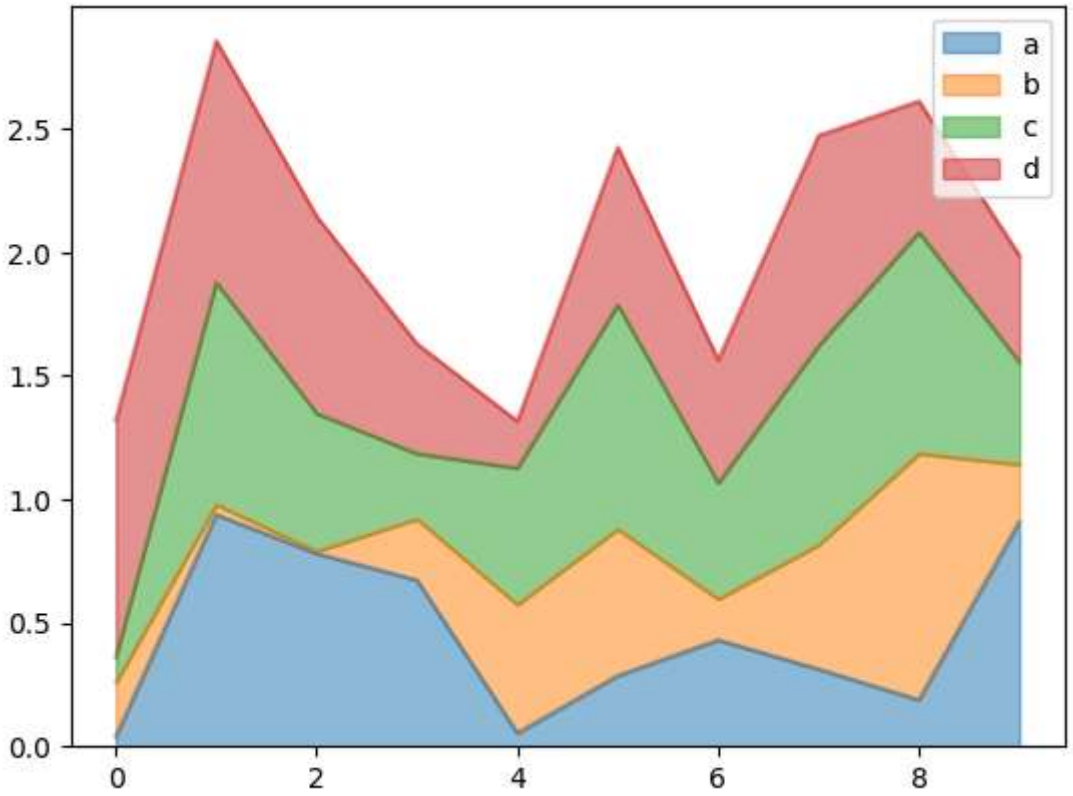
2. Area Plots using Pandas DataFrame

Area plot shows data with a line and fills the space below the line with color.

It helps see how things change over time. we can plot it using DataFrame.plot.area() function.

In [15]:
```
df2.plot.area(alpha=0.5)
```

Out[15]: <Axes: >

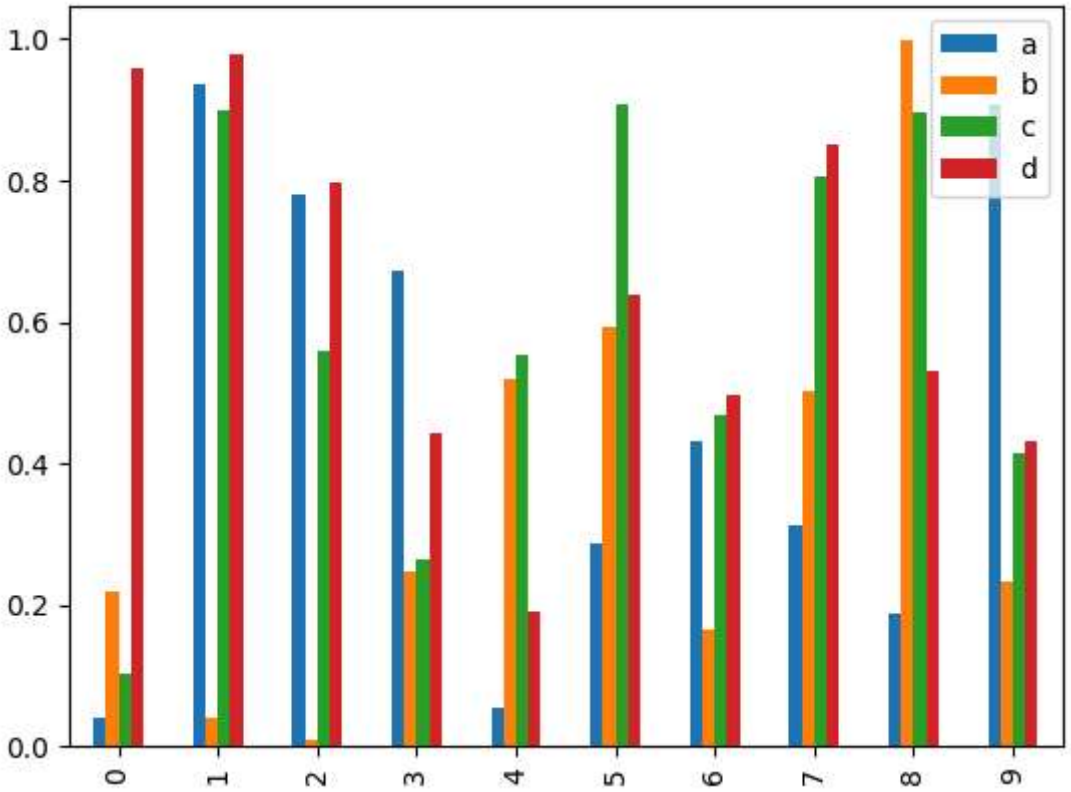


3. Bar Plots using Pandas DataFrame

A bar chart presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally with DataFrame.plot.bar() function.

In [18]:
```
df2.plot.bar()
```

Out[18]: <Axes: >



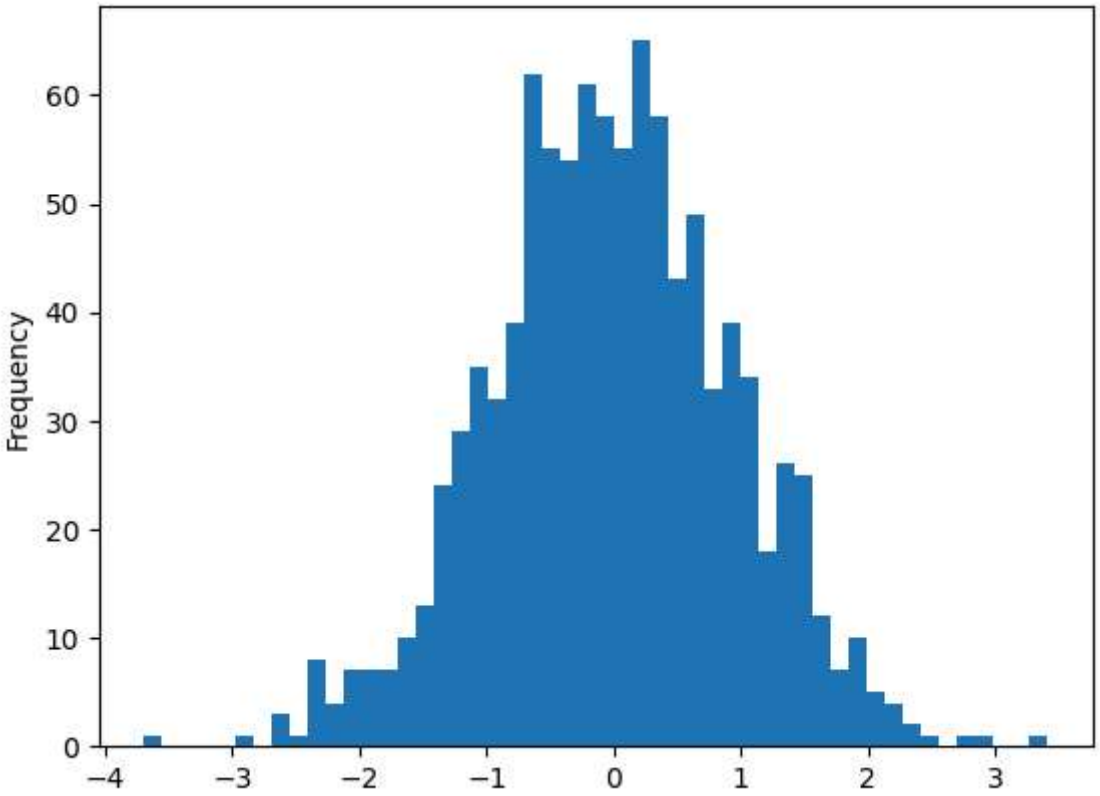
4. Histogram Plot using Pandas DataFrame

Histograms help visualize the distribution of data by grouping values into bins.

Pandas use DataFrame.plot.hist() function to plot histogram.

In [21]:
```
df1['A'].plot.hist(bins=50) # bin- data distribution
```

Out[21]:
```
<Axes: ylabel='Frequency'>
```



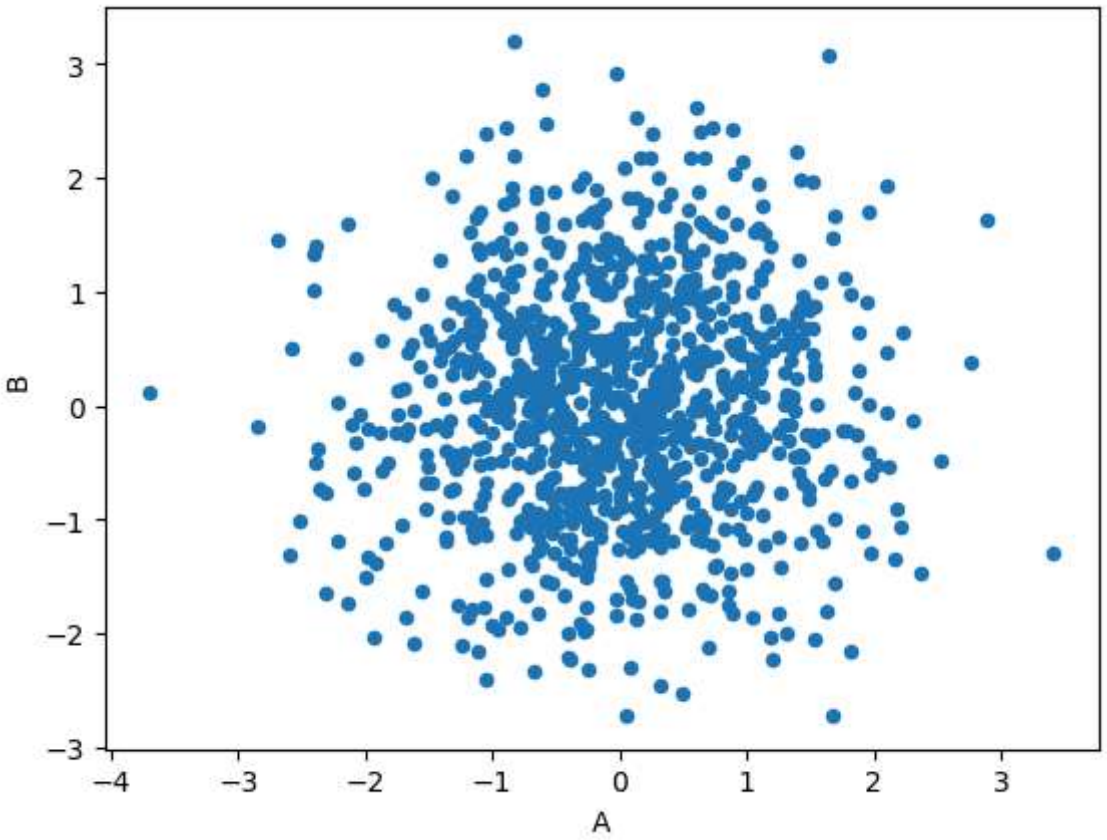
5. Scatter Plot using Pandas DataFrame

Scatter plots are used when you want to show the relationship between two variables.

They are also called correlation and can be created using DataFrame.plot.scatter() function.

In [24]:
```
df1.plot.scatter(x ='A', y ='B')
```

Out[24]: <Axes: xlabel='A', ylabel='B'>



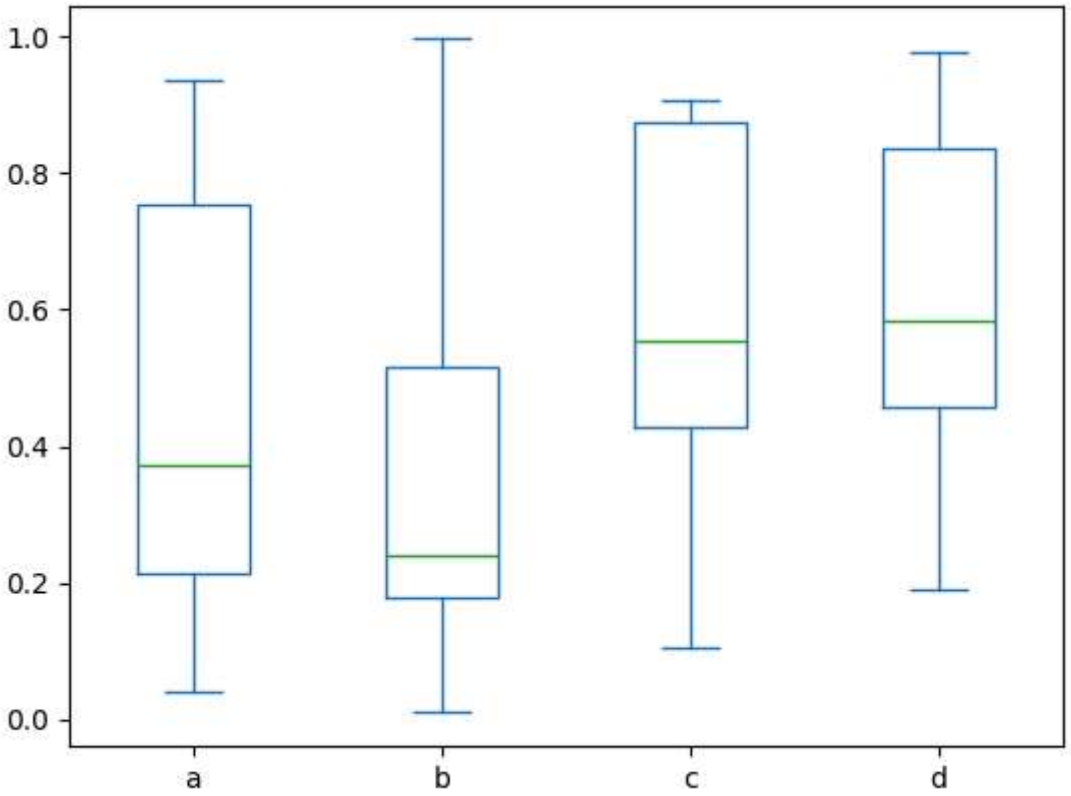
6. Box Plots using Pandas DataFrame

A box plot displays the distribution of data, showing the median, quartiles, and outliers.

we can use DataFrame.plot.box() function or DataFrame.boxplot() to create it.

In [27]:
```
df2.plot.box()
```

Out[27]:
```
<Axes: >
```



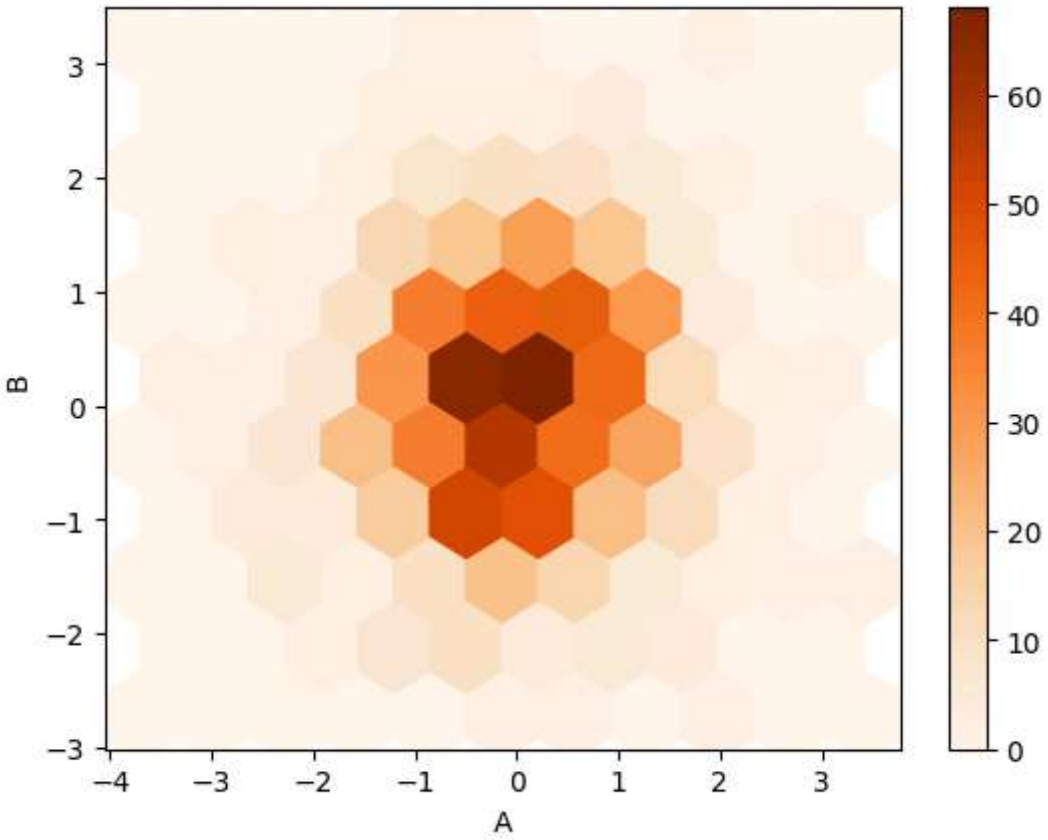
7. Hexagonal Bin Plots using Pandas DataFrame

Hexagonal binning helps manage dense datasets by using hexagons instead of individual points. It's useful for visualizing large datasets where points may overlap.

Let's create the hexagonal bin plot.

In [42]:
```
df1.plot.hexbin(x ='A', y ='B', gridsize = 10, cmap ='Oranges')
```

Out[42]: <Axes: xlabel='A', ylabel='B'>

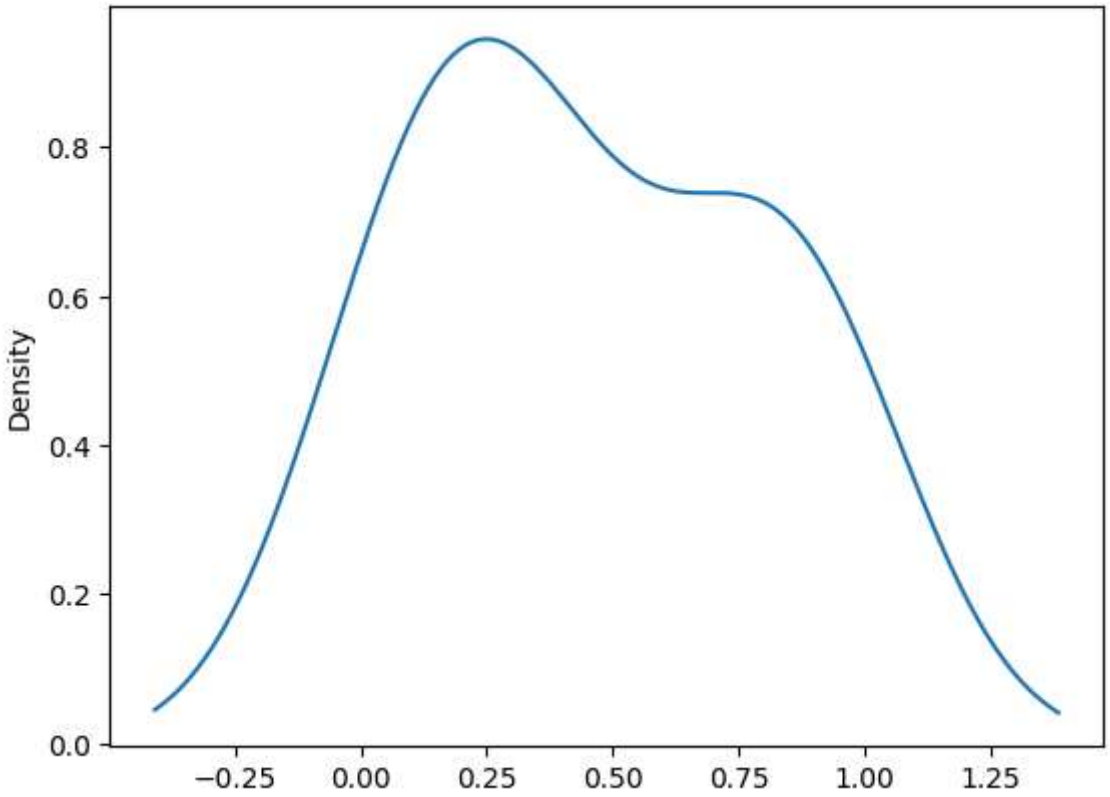


8. Kernel Density Estimation plot (KDE) using Pandas DataFrame

KDE (Kernel Density Estimation) creates a smooth curve to show the shape of data by using the df.plot.kde() function. It's useful for visualizing data patterns and simulating new data based on real examples.

In [45]:
```
df2['a'].plot.kde()
```

Out[45]: <Axes: ylabel='Density'>

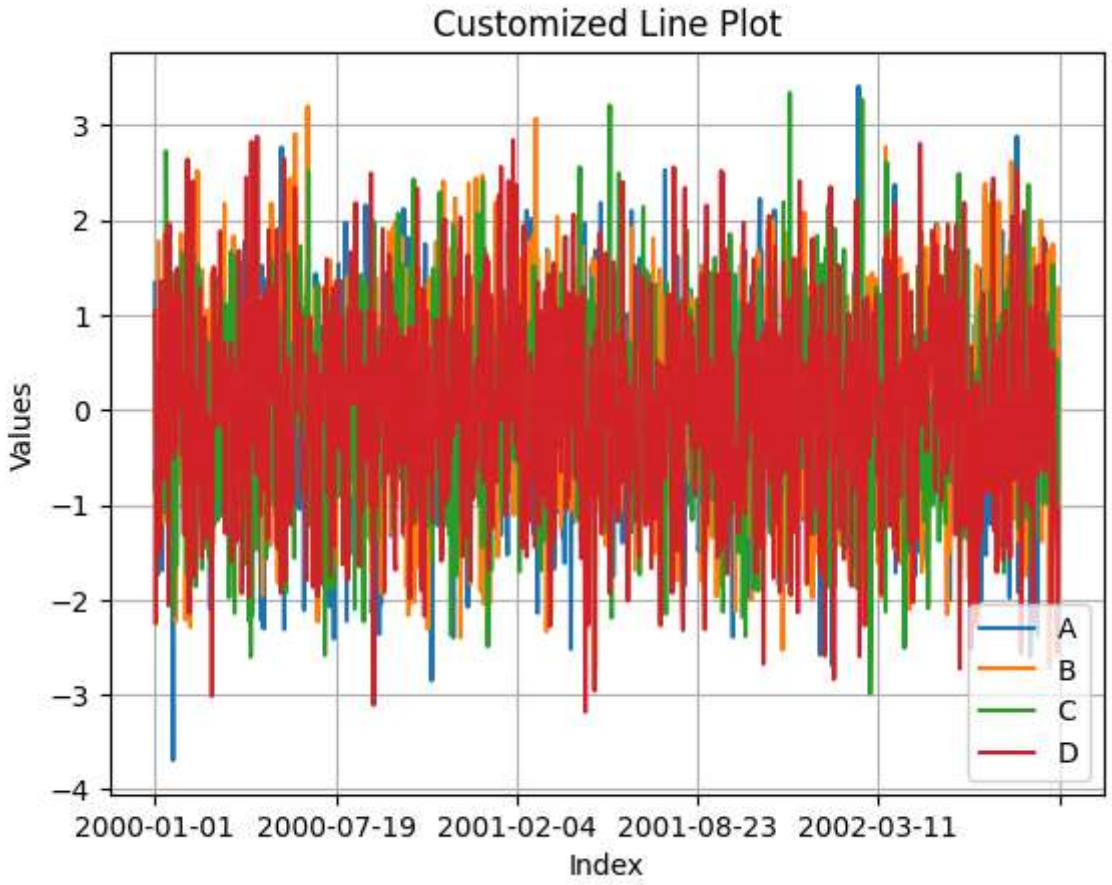


1. Adding a Title, Axis Labels and Gridlines

You can customize the plot by adding a title and labels for the x and y axes. You can also enable gridlines to make the plot easier to read:

In [58]:
```
df1.plot(title='Customized Line Plot', xlabel='Index', ylabel='Values', grid=True)
```

Out[58]:
```
<Axes: title={'center': 'Customized Line Plot'}, xlabel='Index', ylabel='Values'>
```

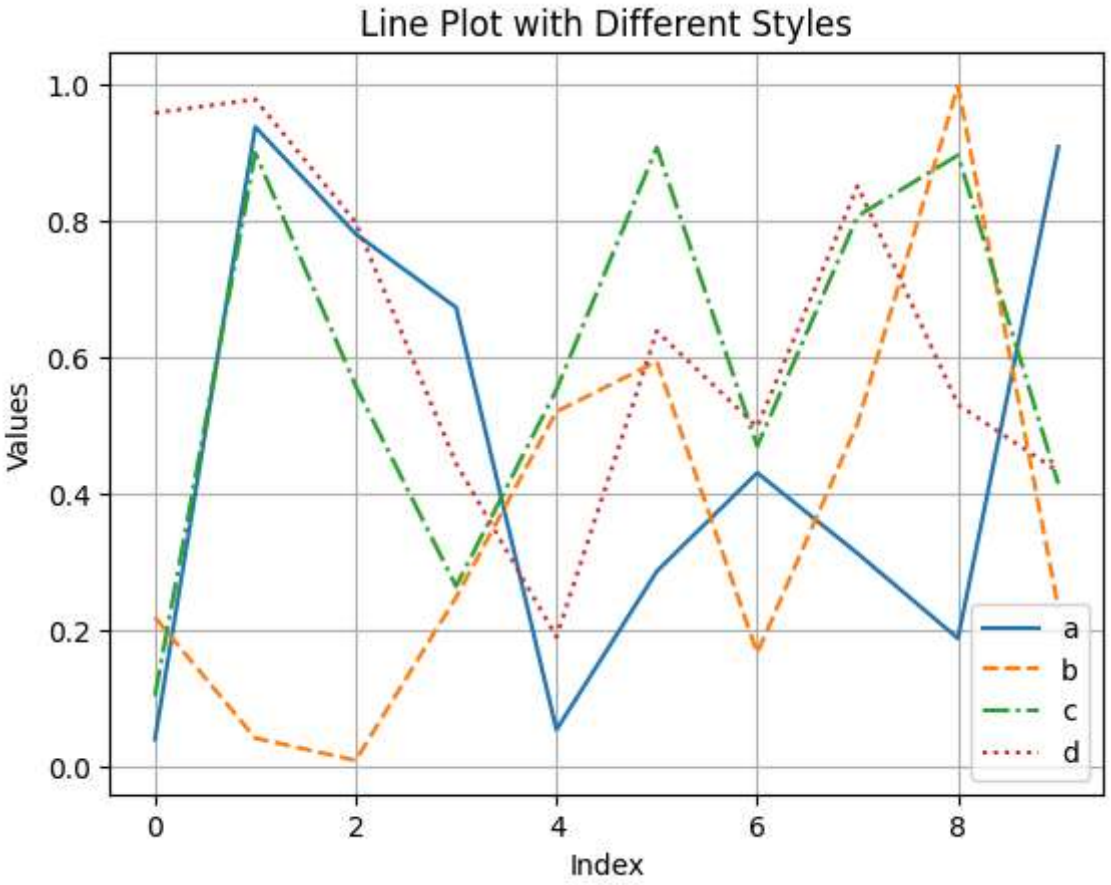


2. Line Plot with Different Line Styles

If you want to differentiate between the two lines visually you can change the line style (e.g., solid line, dashed line) with the help of pandas.

```
In [65]: df2.plot(style=['-', '--', '-.', ':'], title='Line Plot with Different Styles', xlabel='Index', ylabel='Values', gri
```

```
Out[65]: <Axes: title={'center': 'Line Plot with Different Styles'}, xlabel='Index', ylabel='Values'>
```

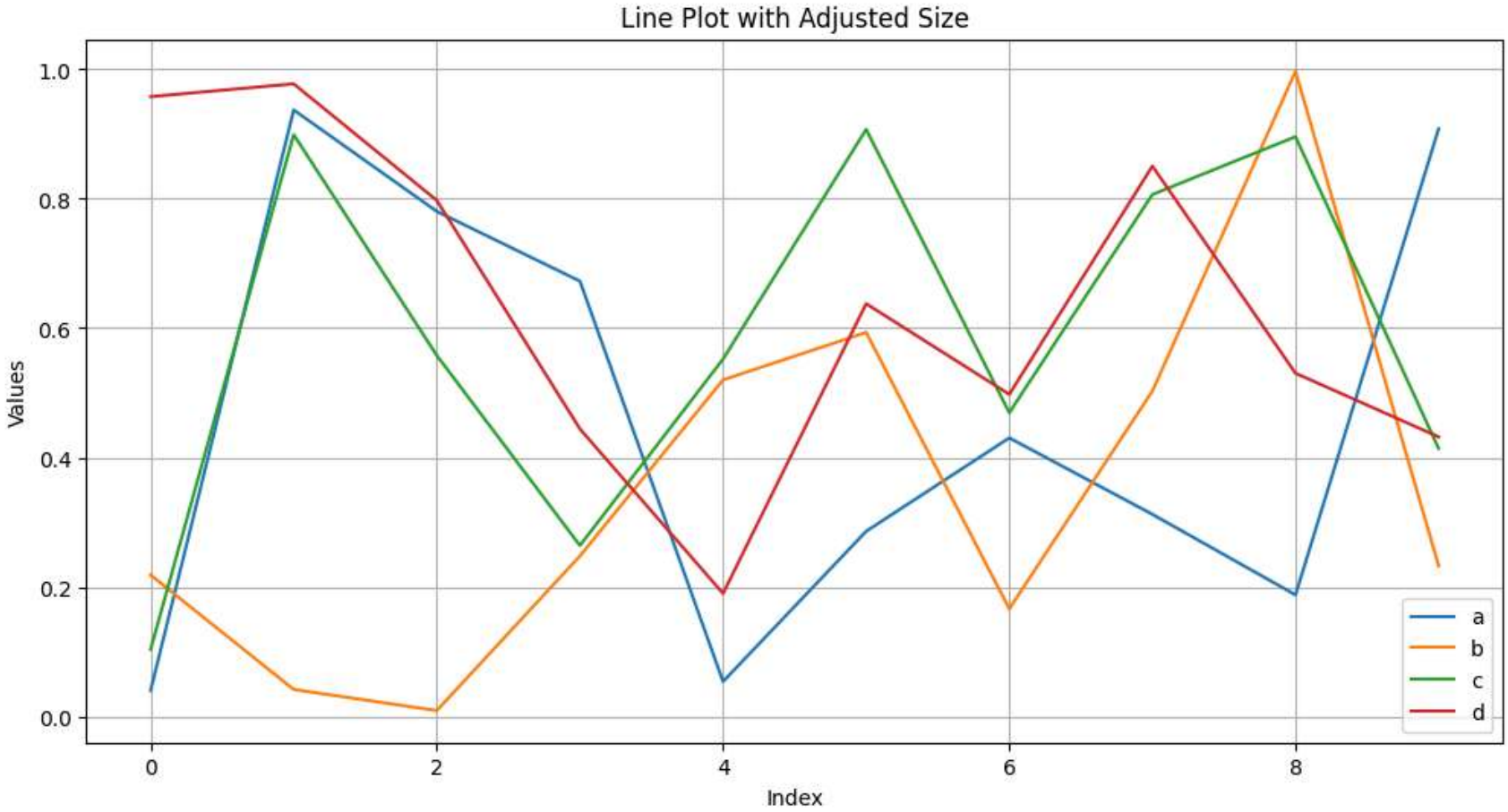


3. Adjusting the Plot Size

Change the size of the plot to better fit the presentation or analysis context You can change it by using the figsize parameter:

In [71]:
```
df2.plot(figsize=(12, 6), title='Line Plot with Adjusted Size', xlabel='Index', ylabel='Values', grid=True)
# width - 12, height -6
```

Out[71]:
```
<Axes: title={'center': 'Line Plot with Adjusted Size'}, xlabel='Index', ylabel='Values'>
```



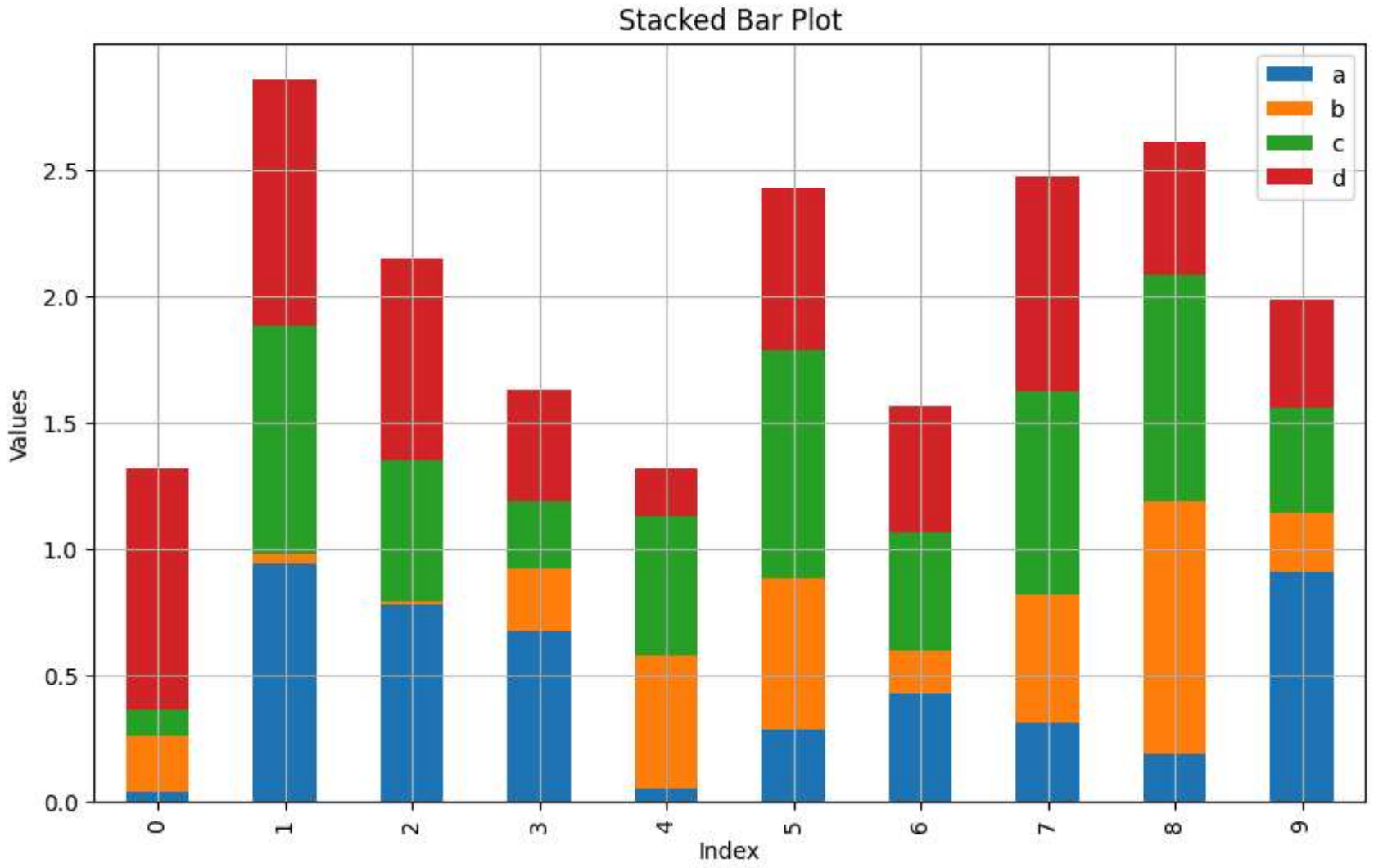
4. Stacked Bar Plot

A stacked bar plot can be created by setting stacked=True.

It helps you visualize the cumulative value for each index.

In [76]:
```
df2.plot.bar(stacked=True, figsize=(10, 6), title='Stacked Bar Plot', xlabel='Index', ylabel='Values', grid=True)
```

Out[76]:
```
<Axes: title={'center': 'Stacked Bar Plot'}, xlabel='Index', ylabel='Values'>
```



In [78]:
```
import pandas as pd

data = {
    'Title': ['RRR', 'Stranger Things', 'Breaking Bad', 'The Mandalorian', 'Avatar: The Last Airbender', 'The Office
    'Genre': ['Drama', 'Sci-Fi', 'Drama', 'Sci-Fi', 'Animation', 'Comedy', 'Fantasy', 'Documentary', 'Comedy', 'Sci-F
    'Release_Year': [2022, 2016, 2008, 2019, 2005, 2005, 2011, 2014, 2016, 2011, 2019, 2013],
    'Director': ['SS RAJ', 'The Duffer Brothers', 'Vince Gilligan', 'Jon Favreau', 'Michael Dante DiMartino, Bryan Ko
    'Seasons': [4, 4, 5, 2, 3, 9, 8, 1, 4, 5, 2, 1],
    'Duration_Minutes': [60, 50, 47, 40, 23, 22, 57, 60, 22, 60, 45, 43]
```

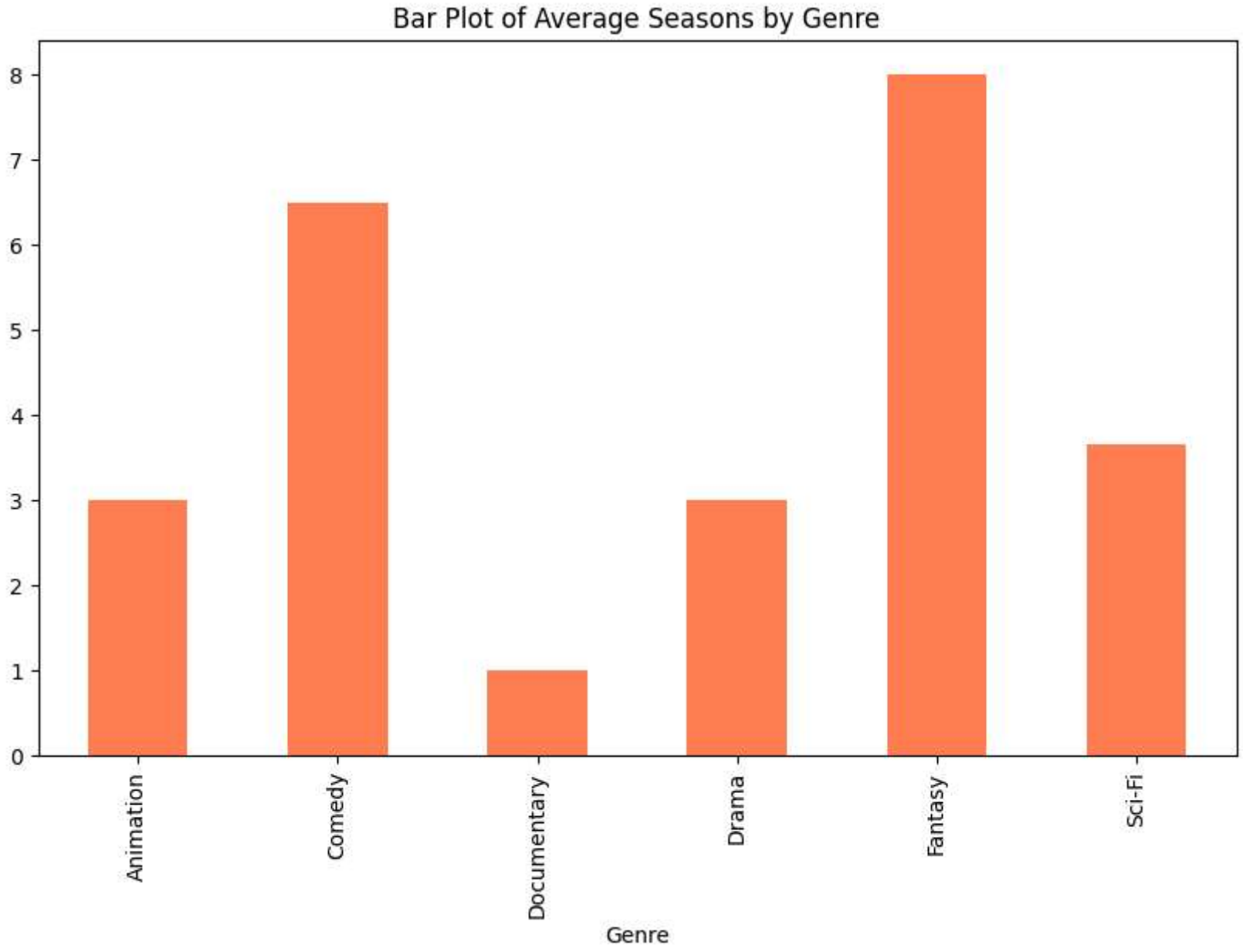
```
}
tv_serials_df = pd.DataFrame(data)
tv_serials_df.head()
```

Out[78]:

	Title	Genre	Release_Year	Director	Seasons	Duration_Minutes
0	RRR	Drama	2022	SS RAJ	4	60
1	Stranger Things	Sci-Fi	2016	The Duffer Brothers	4	50
2	Breaking Bad	Drama	2008	Vince Gilligan	5	47
3	The Mandalorian	Sci-Fi	2019	Jon Favreau	2	40
4	Avatar: The Last Airbender	Animation	2005	Michael Dante DiMartino, Bryan Konietzko	3	23

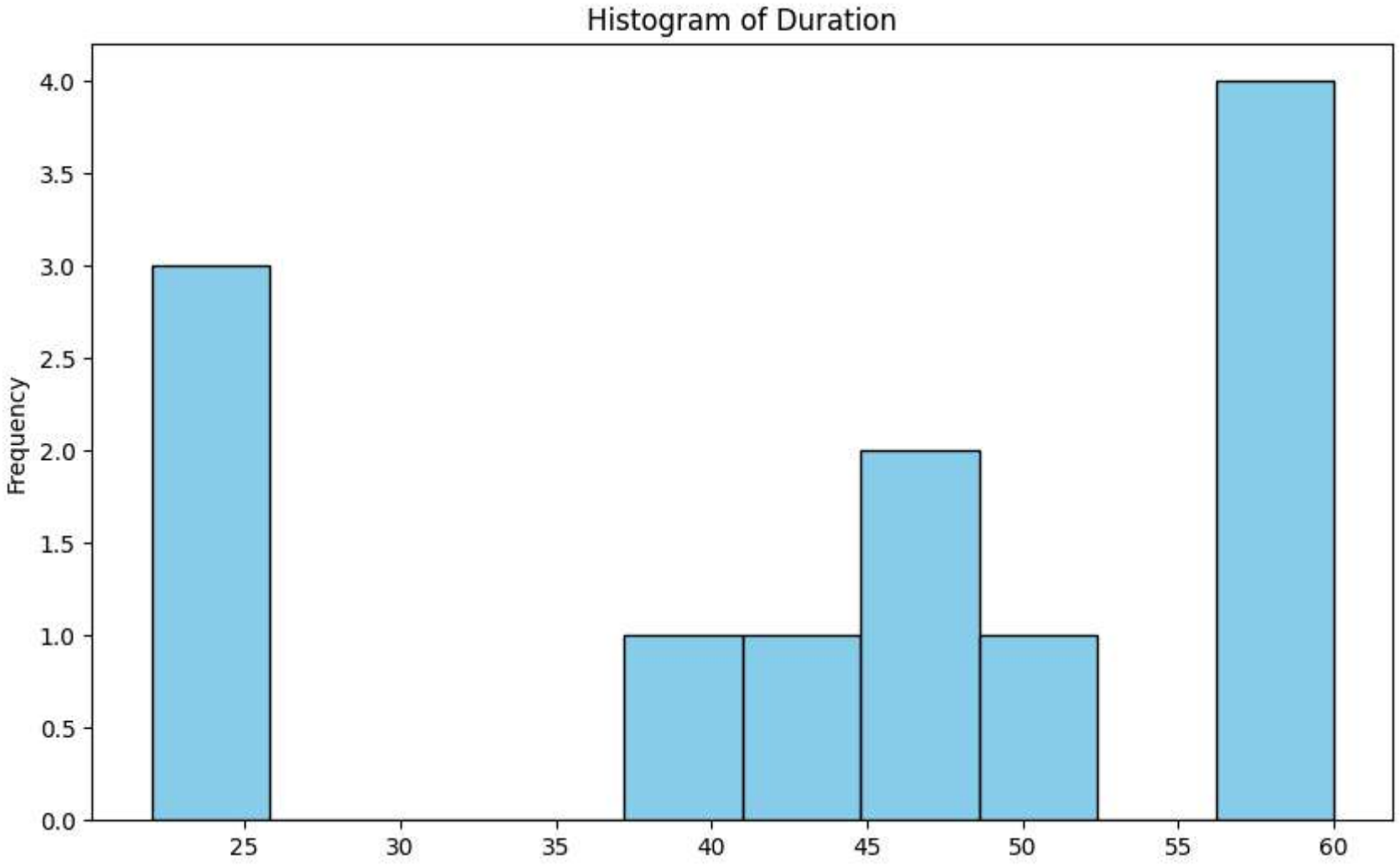
In [84]:
```
genre_seasons = tv_serials_df.groupby('Genre')['Seasons'].mean()
genre_seasons.plot.bar(figsize=(10, 6), color='coral', title='Bar Plot of Average Seasons by Genre')
```

Out[84]:
```
<Axes: title={'center': 'Bar Plot of Average Seasons by Genre'}, xlabel='Genre'>
```



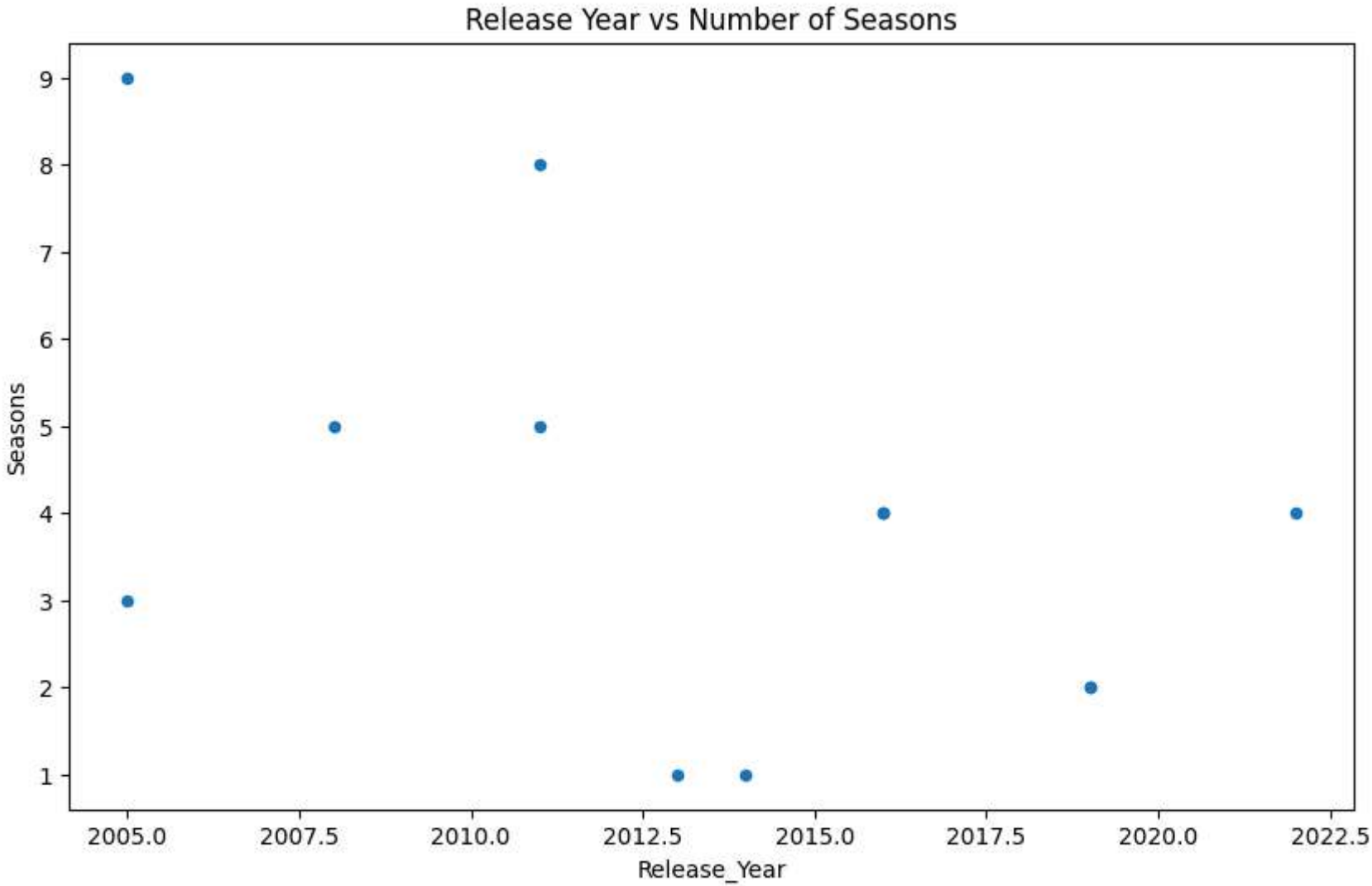
In [90]:
```
tv_serials_df['Duration_Minutes'].plot.hist(bins=10, figsize=(10, 6), color='skyblue', edgecolor='black', title='Hist
```

Out[90]:
```
<Axes: title={'center': 'Histogram of Duration'}, ylabel='Frequency'>
```



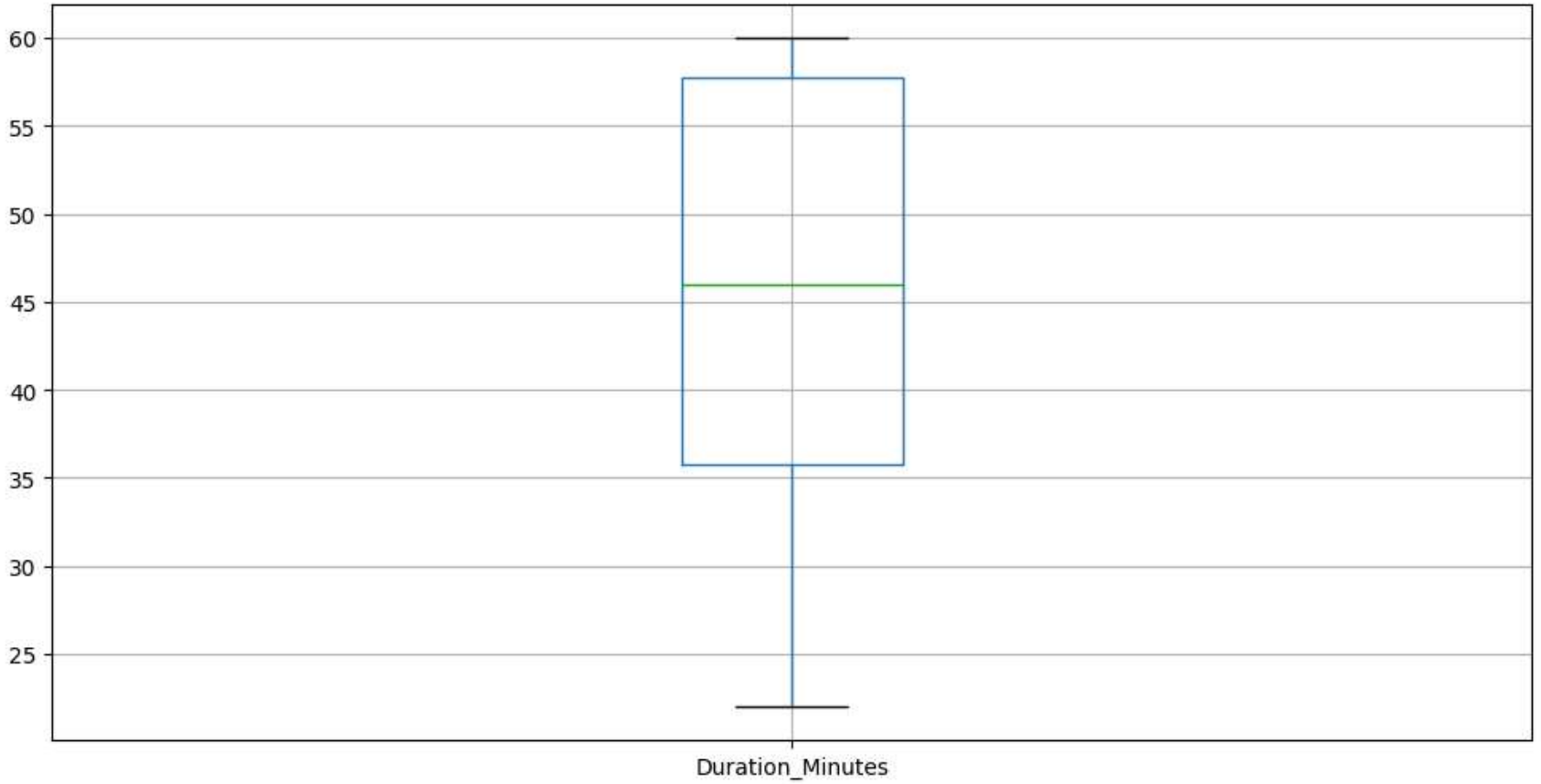
In [92]:
```
# Scatter Plot: Release_Year vs Seasons
tv_serials_df.plot.scatter(x='Release_Year', y='Seasons', figsize=(10, 6), title='Release Year vs Number of Seasons')
```

Out[92]:
```
<Axes: title={'center': 'Release Year vs Number of Seasons'}, xlabel='Release_Year', ylabel='Seasons'>
```



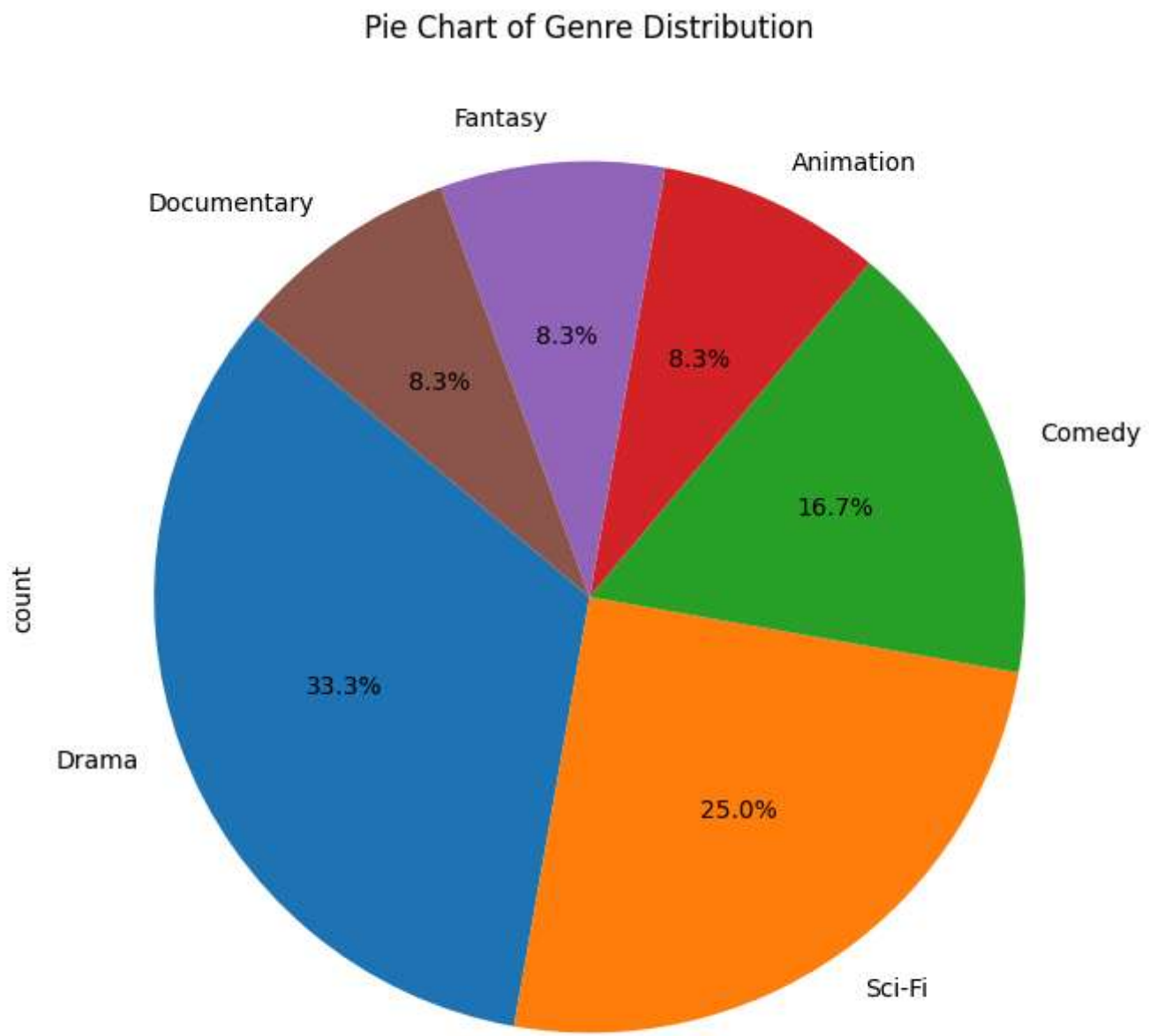
In [98]:
```
# Box Plot Duration_Minutes
tv_serials_df.boxplot(column='Duration_Minutes', figsize=(12, 6), showfliers=False)
```

Out[98]: <Axes: >



In [104…
```
genre_distribution = tv_serials_df['Genre'].value_counts()
genre_distribution.plot(kind='pie', figsize=(8, 8), autopct='%1.1f%%', startangle=140, title='Pie Chart of Genre Dist
```

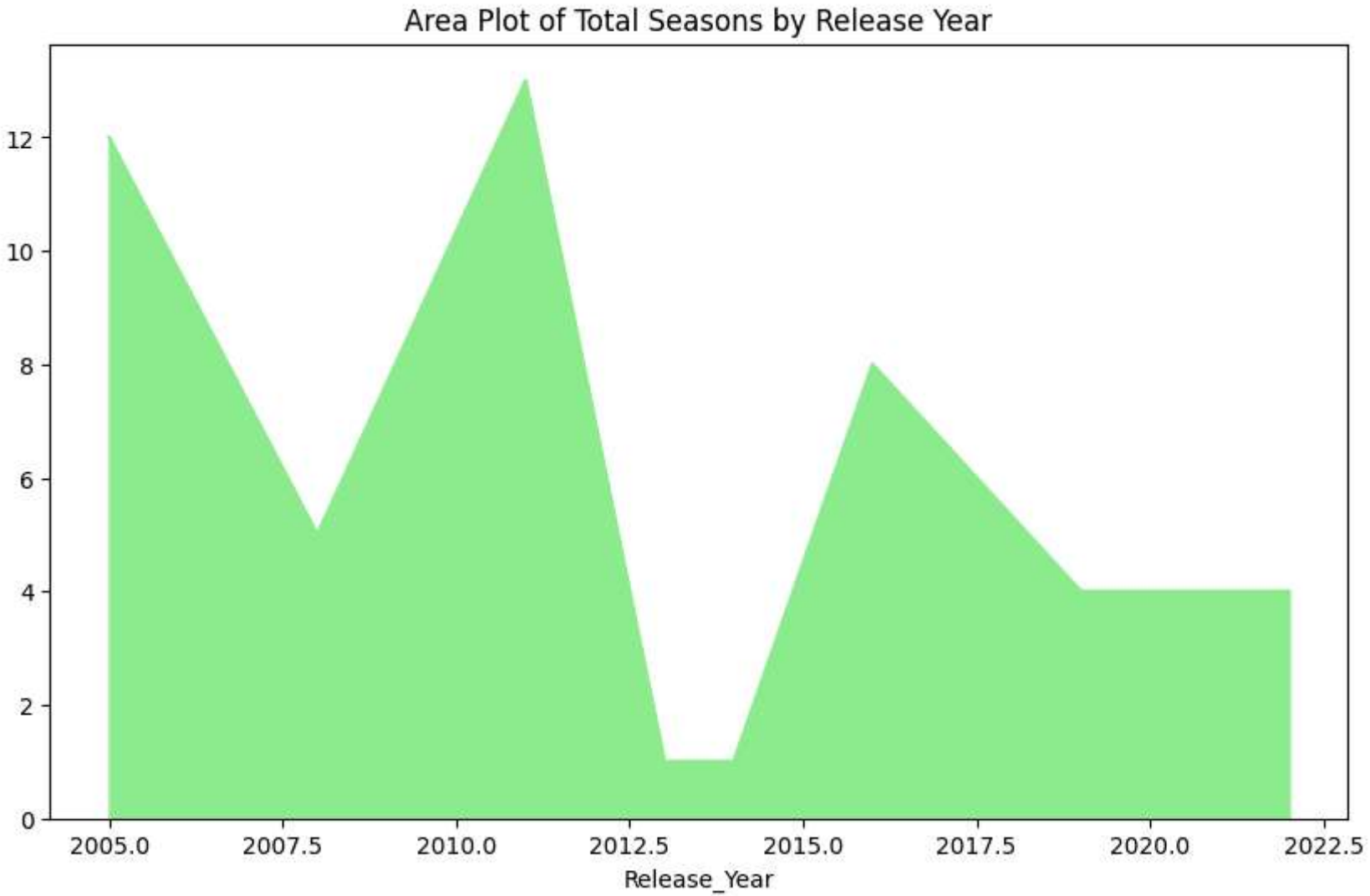
Out[104…
```
<Axes: title={'center': 'Pie Chart of Genre Distribution'}, ylabel='count'>
```



In [106…

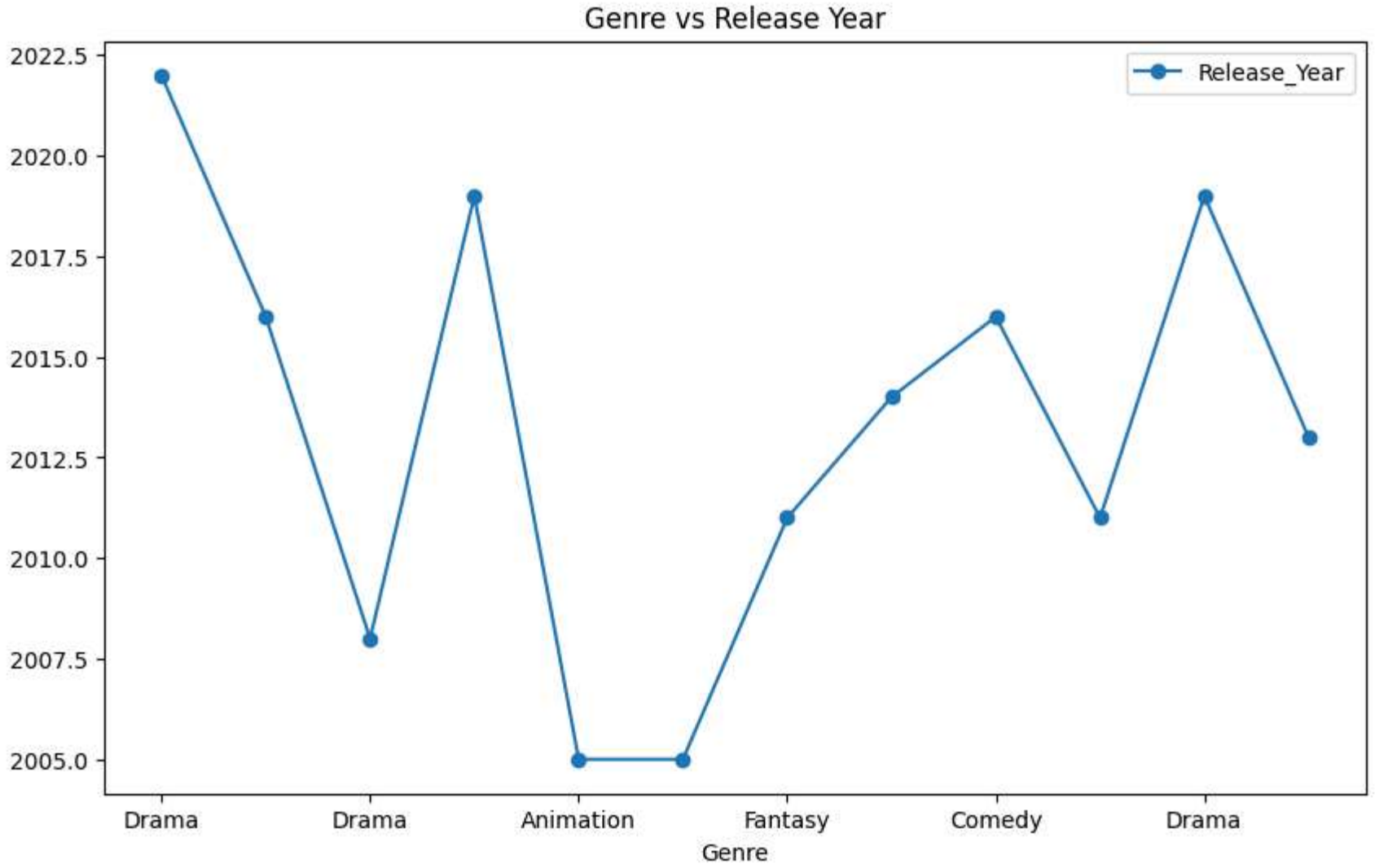
```
tv_serials_df.groupby('Release_Year')['Seasons'].sum().plot(kind='area', figsize=(10, 6), color='lightgreen', title=
```

Out[106… <Axes: title={'center': 'Area Plot of Total Seasons by Release Year'}, xlabel='Release_Year'>



In [108…
```
tv_serials_df.plot.line(x='Genre', y='Release_Year', figsize=(10, 6), title='Genre vs Release Year', marker='o', line
```

Out[108… <Axes: title={'center': 'Genre vs Release Year'}, xlabel='Genre'>


Genre vs Release Year
Release_Year
2022.5
2020.0
2017.5
2015.0
2012.5
2010.0
2007.5
2005.0
Drama
Drama
Animation
Fantasy
Comedy
Drama
Genre

In [ ]: