IMPORTING CSV FILE

In [4]:
```
# import pandas as pd
import pandas as pd
df = pd.read_csv (r"C:\Users\Anusha V\Desktop\YouTube.csv")
print(df)
```

```
     Rank             Username  Categories  Suscribers         Country  \
0       1              tseries        True   249500000           India
1       2              MrBeast       False   183500000  Estados Unidos
2       3            CoComelon        True   165500000         Unknown
3       4              SETIndia       False   162600000           India
4       5         KidsDianaShow        True   113500000         Unknown
..    ...                  ...         ...         ...             ...
194   195  dianaandromahin8102        True    26200000         Unknown
195   196           nickiminaj       False    26100000  Estados Unidos
196   197  mariliamendoncareal        True    26100000          Brasil
197   198         TheLallantop       False    26000000           India
198   199                  AGT        True    25900000  Estados Unidos

        Visits    Likes  Comments  \
0        86200     2700        78
1    117400000  5300000     18500
2      7000000    24700         0
3        15600      166         9
4      3900000    12400         0
..         ...      ...       ...
194     109000      322         0
195    1600000    98300      7600
196          0        0         0
197      30800      822        52
198     186800     3100       197

                                                 Links
0    http://youtube.com/channel/UCq-Fj5jknLsUf-MWSy...  (http://youtube.co
m/channel/UCq-Fj5jknLsUf-MWSy...)
1    http://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u...  (http://youtube.co
m/channel/UCX6OQ3DkcsbYNE6H8u...)
2    http://youtube.com/channel/UCbCmjCuTUZos6Inko4...  (http://youtube.co
m/channel/UCbCmjCuTUZos6Inko4...)
3    http://youtube.com/channel/UCpEhnqL0y41EpW2TvW...  (http://youtube.co
m/channel/UCpEhnqL0y41EpW2TvW...)
4    http://youtube.com/channel/UCk8GzjMOrta8yxDcKf...  (http://youtube.co
m/channel/UCk8GzjMOrta8yxDcKf...)
..                                                 ...
194  http://youtube.com/channel/UC5ma-WCc8jNCV0WyRn...  (http://youtube.co
m/channel/UC5ma-WCc8jNCV0WyRn...)
195  http://youtube.com/channel/UC3jOd7GUMhpgJRBhiL...  (http://youtube.co
m/channel/UC3jOd7GUMhpgJRBhiL...)
196  http://youtube.com/channel/UCwfEOn0O1DWcyTgzVV...  (http://youtube.co
m/channel/UCwfEOn0O1DWcyTgzVV...)
197  http://youtube.com/channel/UCx8Z14PpntdaxCt2ha...  (http://youtube.co
m/channel/UCx8Z14PpntdaxCt2ha...)
198  http://youtube.com/channel/UCT2X19JJaJGUN7mrYu...  (http://youtube.co
m/channel/UCT2X19JJaJGUN7mrYu...)

[199 rows x 9 columns]
```

HEAD FUNCTION

In [288]:
```
1 df.head(20)
```

Out[288]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comn
0	1	tseries	True	249500000	India	86200	2700	
1	2	MrBeast	False	183500000	Estados Unidos	117400000	5300000	1
2	3	CoComelon	True	165500000	Unknown	7000000	24700	
3	4	SETIndia	False	162600000	India	15600	166	
4	5	KidsDianaShow	True	113500000	Unknown	3900000	12400	
5	6	PewDiePie	False	111500000	Estados Unidos	2400000	197300	
6	7	LikeNastyaofficial	True	107500000	Unknown	2600000	28000	
7	8	VladandNiki	False	101400000	Unknown	4100000	22100	
8	9	zeemusiccompany	True	99700000	India	74300	2600	
9	10	WWE	False	97200000	Estados Unidos	184500	6300	
10	11	BLACKPINK	True	91300000	Estados Unidos	863200	146900	
11	12	GoldminesTelefilms	False	89700000	India	34600	421	
12	13	SonySAB	True	85400000	India	35500	615	
13	14	5MinuteCraftsYouTube	False	80300000	India	164600	703	
14	15	BTS	True	76500000	India	969700	180300	
15	16	HYBELABELS	False	76500001	India	969700	180300	
16	17	HYBELABELS	True	76500002	India	969700	180300	
17	18	Pinkfong	False	69600000	Unknown	506400	397	
18	19	ChuChuTV	True	67500000	Unknown	652100	17800	
19	20	KondZilla	False	66600000	Brasil	30400	1100	

TAIL FUNCTION

In [289]:
```
1 df.head(20)
```

Out[289]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comn
0	1	tseries	True	249500000	India	86200	2700	
1	2	MrBeast	False	183500000	Estados Unidos	117400000	5300000	1
2	3	CoComelon	True	165500000	Unknown	7000000	24700	
3	4	SETIndia	False	162600000	India	15600	166	
4	5	KidsDianaShow	True	113500000	Unknown	3900000	12400	
5	6	PewDiePie	False	111500000	Estados Unidos	2400000	197300	
6	7	LikeNastyaofficial	True	107500000	Unknown	2600000	28000	
7	8	VladandNiki	False	101400000	Unknown	4100000	22100	
8	9	zeemusiccompany	True	99700000	India	74300	2600	
9	10	WWE	False	97200000	Estados Unidos	184500	6300	
10	11	BLACKPINK	True	91300000	Estados Unidos	863200	146900	
11	12	GoldminesTelefilms	False	89700000	India	34600	421	
12	13	SonySAB	True	85400000	India	35500	615	
13	14	5MinuteCraftsYouTube	False	80300000	India	164600	703	
14	15	BTS	True	76500000	India	969700	180300	
15	16	HYBELABELS	False	76500001	India	969700	180300	
16	17	HYBELABELS	True	76500002	India	969700	180300	
17	18	Pinkfong	False	69600000	Unknown	506400	397	
18	19	ChuChuTV	True	67500000	Unknown	652100	17800	
19	20	KondZilla	False	66600000	Brasil	30400	1100	

DESCRIBE FUNCTION

In [290]:
```
1 df.describe()
```

Out[290]:

	Rank	Suscribers	Visits	Likes	Comments
count	199.000000	1.990000e+02	1.990000e+02	1.990000e+02	199.000000
mean	100.000000	4.405226e+07	2.261227e+06	1.047747e+05	2096.356784
std	57.590508	2.753718e+07	1.084192e+07	5.321669e+05	7913.381544
min	1.000000	2.590000e+07	0.000000e+00	0.000000e+00	0.000000
25%	50.500000	3.030000e+07	3.225000e+04	5.845000e+02	0.000000
50%	100.000000	3.550000e+07	2.257000e+05	3.900000e+03	28.000000
75%	149.500000	4.500000e+07	1.050000e+06	3.465000e+04	375.000000
max	199.000000	2.495000e+08	1.174000e+08	5.300000e+06	82800.000000

PIVOT FUNCTION

In [227]:
```
pivot_table = df.pivot_table(index = 'Likes', aggfunc='mean')
pivot_table
```

Out[227]:

Likes	Categories	Rank
0	4567.0	124.125
1000	4567.0	161.000
10100	4567.0	70.000
10200	4567.0	36.000
106500	4567.0	157.000
...	...	...
983	4567.0	23.000
98300	4567.0	196.000
993	4567.0	192.000
994	4567.0	64.000
HYBELABELS	4567.0	16.500

167 rows × 2 columns

MELT FUNCTION

In [228]:
```
melt_df = pd.melt(df,id_vars=['Likes'], var_name='visits')
melt_df
```

Out[228]:

	Likes	visits	value
0	2700	Rank	1
1	5300000	Rank	2
2	24700	Rank	3
3	166	Rank	4
4	12400	Rank	5
...	...	...	...
2383	322	Unnamed: 12	NaN
2384	98300	Unnamed: 12	NaN
2385	0	Unnamed: 12	NaN
2386	822	Unnamed: 12	NaN
2387	3100	Unnamed: 12	NaN

2388 rows × 3 columns

REDUCE FUNCTION

In [175]:

```
from functools import reduce
def multiply(x, y):
    return x * y
column_to_reduce = 'Rank'
result = reduce(multiply, df['Rank'])
print(result)
```

```
394328933682395251776181606966092531147567988843586631647371266622179724981701671460152142005992311952088606069459819415128821395121318552530963312476414965556731428635381658618698494471961222810725832120127016645932065613714147426638762121203786951620160628702789784330113015952085162031175850429398089461111394811851948687360000000000000000000000000000000000000000000000000000
```

MAPPING FUNCTION

In [5]:

```
def map_grades(grade):
    grade_maping = {
        'A':90,
        'B':80,
        'C':70,
        'D':60,
        'E':50,
        'F':40
}
LeagueIndex = 'Grade'
df['Likes'] = df['Rank'].map(map_grades)
print(df)
```

```
     Rank             Username  Categories  Suscribers         Country  \
0       1              tseries        True   249500000           India   
1       2              MrBeast       False   183500000  Estados Unidos   
2       3            CoComelon        True   165500000         Unknown   
3       4              SETIndia       False   162600000           India   
4       5         KidsDianaShow        True   113500000         Unknown   
..    ...                  ...         ...         ...             ...   
194   195  dianaandromahin8102        True    26200000         Unknown   
195   196           nickiminaj       False    26100000  Estados Unidos   
196   197  mariliamendoncareal        True    26100000          Brasil   
197   198         TheLallantop       False    26000000           India   
198   199                  AGT        True    25900000  Estados Unidos   

        Visits Likes  Comments  \
0        86200  None        78   
1    117400000  None     18500   
2      7000000  None         0   
3        15600  None         9   
4      3900000  None         0   
..         ...   ...       ...   
194     109000  None         0   
195    1600000  None      7600   
196          0  None         0   
197      30800  None        52   
198     186800  None       197   

                                                 Links  
0    http://youtube.com/channel/UCq-Fj5jknLsUf-MWSy...  
1    http://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u...  
2    http://youtube.com/channel/UCbCmjCuTUZos6Inko4...  
3    http://youtube.com/channel/UCpEhnqL0y41EpW2TvW...  
4    http://youtube.com/channel/UCk8GzjMOrta8yxDcKf...  
..                                                 ...  
194  http://youtube.com/channel/UC5ma-WCc8jNCV0WyRn...  
195  http://youtube.com/channel/UC3jOd7GUMhpgJRBhiL...  
196  http://youtube.com/channel/UCwfEOn0O1DWcyTgzVV...  
197  http://youtube.com/channel/UCx8Z14PpntdaxCt2ha...  
198  http://youtube.com/channel/UCT2X19JJaJGUN7mrYu...  

[199 rows x 9 columns]
```

Filter FUNCTION

In [8]:
```
1 def score_condition(score):
2     return score > 5
3 Likes = 'score_condition'
4 filtered_df = df[df['Likes'].apply(score_condition)]
5 print(filtered_df)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_10508\466627290.py in <module>
      2     return score > 5
      3 Likes = 'score_condition'
----> 4 filtered_df = df[df['Likes'].apply(score_condition)]
      5 print(filtered_df)

~\anaconda3\lib\site-packages\pandas\core\series.py in apply(self, func, c
onvert_dtype, args, **kwargs)
   4431         dtype: float64
   4432         """
-> 4433         return SeriesApply(self, func, convert_dtype, args, kwarg
s).apply()
   4434
   4435     def _reduce(

~\anaconda3\lib\site-packages\pandas\core\apply.py in apply(self)
   1086             return self.apply_str()
   1087
-> 1088         return self.apply_standard()
   1089
   1090     def agg(self):

~\anaconda3\lib\site-packages\pandas\core\apply.py in apply_standard(self)
   1141                 # List[Union[Callable[..., Any], str]]]]"; expect
ed
   1142                 # "Callable[[Any], Any]"
-> 1143                 mapped = lib.map_infer(
   1144                     values,
   1145                     f,  # type: ignore[arg-type]

~\anaconda3\lib\site-packages\pandas\_libs\lib.pyx in pandas._libs.lib.map
_infer()

~\AppData\Local\Temp\ipykernel_10508\466627290.py in score_condition(scor
e)
      1 def score_condition(score):
----> 2     return score > 5
      3 Likes = 'score_condition'
      4 filtered_df = df[df['Likes'].apply(score_condition)]
      5 print(filtered_df)

TypeError: '>' not supported between instances of 'NoneType' and 'int'
```

LAMBDA FUNCTION

In [178]:

```
modify_score = lambda x: x * 2
LeagueIndex = 'score'
df['Likes'] = df['Rank'].apply(modify_score)
print(df)
```

	Rank	Username	Categories	Suscribers	\
0	1	tseries	MÃºsica y baile	249500000	
1	2	MrBeast	Videojuegos, Humor	183500000	
2	3	CoComelon	EducaciÃ³n	165500000	
3	4	SETIndia	EducaciÃ³n	162600000	
4	5	KidsDianaShow	AnimaciÃ³n, Juguetes	113500000	
..	...	...	...	...	
194	195	dianaandromahin8102	AnimaciÃ³n, Juguetes	26200000	
195	196	nickiminaj	MÃºsica y baile	26100000	
196	197	mariliamendoncareal	MÃºsica y baile	26100000	
197	198	TheLallantop	Noticias y PolÃtica	26000000	
198	199	AGT	MÃºsica y baile, PelÃculas	25900000	

	Country	Visits	Likes	Comments	\
0	India	86200	2	78	
1	Estados Unidos	117400000	4	18500	
2	Unknown	7000000	6	0	
3	India	15600	8	9	
4	Unknown	3900000	10	0	
..	...	...	...	...	
194	Unknown	109000	390	0	
195	Estados Unidos	1600000	392	7600	
196	Brasil	0	394	0	
197	India	30800	396	52	
198	Estados Unidos	186800	398	197	

	Links	Unnamed: 9	Unnamed: 10	\
0	http://youtube.com/channel/UCq-Fj5jknLsUf-MWSy... (http://youtube.com/channel/UCq-Fj5jknLsUf-MWSy...)	NaN	NaN	
1	http://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u... (http://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u...)	NaN	NaN	
2	http://youtube.com/channel/UCbCmjCuTUZos6Inko4... (http://youtube.com/channel/UCbCmjCuTUZos6Inko4...)	NaN	NaN	
3	http://youtube.com/channel/UCpEhnqL0y41EpW2TvW... (http://youtube.com/channel/UCpEhnqL0y41EpW2TvW...)	NaN	NaN	
4	http://youtube.com/channel/UCk8GzjMOrta8yxDcKf... (http://youtube.com/channel/UCk8GzjMOrta8yxDcKf...)	NaN	NaN	
..	...	...	...	
194	http://youtube.com/channel/UC5ma-WCc8jNCV0WyRn... (http://youtube.com/channel/UC5ma-WCc8jNCV0WyRn...)	NaN	NaN	
195	http://youtube.com/channel/UC3jOd7GUMhpgJRBhiL... (http://youtube.com/channel/UC3jOd7GUMhpgJRBhiL...)	NaN	NaN	
196	http://youtube.com/channel/UCwfEOn0O1DWcyTgzVV... (http://youtube.com/channel/UCwfEOn0O1DWcyTgzVV...)	NaN	NaN	
197	http://youtube.com/channel/UCx8Z14PpntdaxCt2ha... (http://youtube.com/channel/UCx8Z14PpntdaxCt2ha...)	NaN	NaN	
198	http://youtube.com/channel/UCT2X19JJaJGUN7mrYu... (http://youtube.com/channel/UCT2X19JJaJGUN7mrYu...)	NaN	NaN	

	Unnamed: 11	Unnamed: 12
0	NaN	NaN
1	NaN	NaN
2	NaN	NaN
3	NaN	NaN
4	NaN	NaN
..	...	...
194	NaN	NaN
195	NaN	NaN
196	NaN	NaN

```
197         NaN         NaN
198         NaN         NaN

[199 rows x 13 columns]
```

SORTED FUNCTION

In [179]:

```
sorted_df = df.sort_values(by='Rank', ascending=False)
print(sorted_df)
```

```
     Rank             Username                   Categories Suscribers  \
198   199                  AGT  MÃºsica y baile, PelÃculas   25900000
197   198          TheLallantop         Noticias y PolÃtica   26000000
196   197  mariliamendoncareal              MÃºsica y baile   26100000
195   196           nickiminaj              MÃºsica y baile   26100000
194   195  dianaandromahin8102         AnimaciÃ³n, Juguetes   26200000
..    ...                  ...                         ...        ...
4       5        KidsDianaShow         AnimaciÃ³n, Juguetes  113500000
3       4              SETIndia                   EducaciÃ³n  162600000
2       3            CoComelon                   EducaciÃ³n  165500000
1       2              MrBeast           Videojuegos, Humor  183500000
0       1              tseries              MÃºsica y baile  249500000

            Country     Visits  Likes Comments  \
198  Estados Unidos     186800    398      197
197           India      30800    396       52
196          Brasil          0    394        0
195  Estados Unidos    1600000    392     7600
194         Unknown     109000    390        0
..              ...        ...    ...      ...
4           Unknown    3900000     10        0
3             India      15600      8        9
2           Unknown    7000000      6        0
1    Estados Unidos  117400000      4    18500
0             India      86200      2       78

                                                    Links Unnamed: 9 Unnamed:
10  \
198  http://youtube.com/channel/UCT2X19JJaJGUN7mrYu... (http://youtube.co
m/channel/UCT2X19JJaJGUN7mrYu...)        NaN         NaN
197  http://youtube.com/channel/UCx8Z14PpntdaxCt2ha... (http://youtube.co
m/channel/UCx8Z14PpntdaxCt2ha...)        NaN         NaN
196  http://youtube.com/channel/UCwfEOn0O1DWcyTgzVV... (http://youtube.co
m/channel/UCwfEOn0O1DWcyTgzVV...)        NaN         NaN
195  http://youtube.com/channel/UC3jOd7GUMhpgJRBhiL... (http://youtube.co
m/channel/UC3jOd7GUMhpgJRBhiL...)        NaN         NaN
194  http://youtube.com/channel/UC5ma-WCc8jNCV0WyRn... (http://youtube.co
m/channel/UC5ma-WCc8jNCV0WyRn...)        NaN         NaN
..                                                   ...        ...
...
4    http://youtube.com/channel/UCk8GzjMOrta8yxDcKf... (http://youtube.co
m/channel/UCk8GzjMOrta8yxDcKf...)        NaN         NaN
3    http://youtube.com/channel/UCpEhnqL0y41EpW2TvW... (http://youtube.co
m/channel/UCpEhnqL0y41EpW2TvW...)        NaN         NaN
2    http://youtube.com/channel/UCbCmjCuTUZos6Inko4... (http://youtube.co
m/channel/UCbCmjCuTUZos6Inko4...)        NaN         NaN
1    http://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u... (http://youtube.co
m/channel/UCX6OQ3DkcsbYNE6H8u...)        NaN         NaN
0    http://youtube.com/channel/UCq-Fj5jknLsUf-MWSy... (http://youtube.co
m/channel/UCq-Fj5jknLsUf-MWSy...)        NaN         NaN

    Unnamed: 11 Unnamed: 12
198         NaN         NaN
197         NaN         NaN
196         NaN         NaN
195         NaN         NaN
194         NaN         NaN
..          ...         ...
4           NaN         NaN
3           NaN         NaN
2           NaN         NaN
```

```
1           NaN         NaN
0           NaN         NaN

[199 rows x 13 columns]
```

GROUPBY FUNCTION

In [180]:
```
grouped_data = df.groupby('Likes')
result = grouped_data['Likes'].mean()
print(result)
```

```
Likes
2        2.0
4        4.0
6        6.0
8        8.0
10      10.0
       ...
390    390.0
392    392.0
394    394.0
396    396.0
398    398.0
Name: Likes, Length: 199, dtype: float64
```

AGGREGATION FUNCTIONS

In [181]:
```
sum_result = df['Likes'].sum()
sum_result
```

Out[181]: 39800

In [182]:
```
sum_result = df['Likes'].min()
sum_result
```

Out[182]: 2

In [183]:
```
sum_result = df['Likes'].median()
sum_result
```

Out[183]: 200.0

In [184]:
```
sum_result = df['Likes'].median()
sum_result
```

Out[184]: 200.0

In [185]:
```
sum_result = df['Likes'].max()
sum_result
```

Out[185]: 398

In [186]:
```
sum_result = df['Likes'].count()
sum_result
```

Out[186]: 199

In [187]:
```
sum_result = df['Likes'].std()
sum_result
```

Out[187]: 115.18101695447331

In [188]:
```
sum_result = df['Likes'].var()
sum_result
```

Out[188]: 13266.666666666666

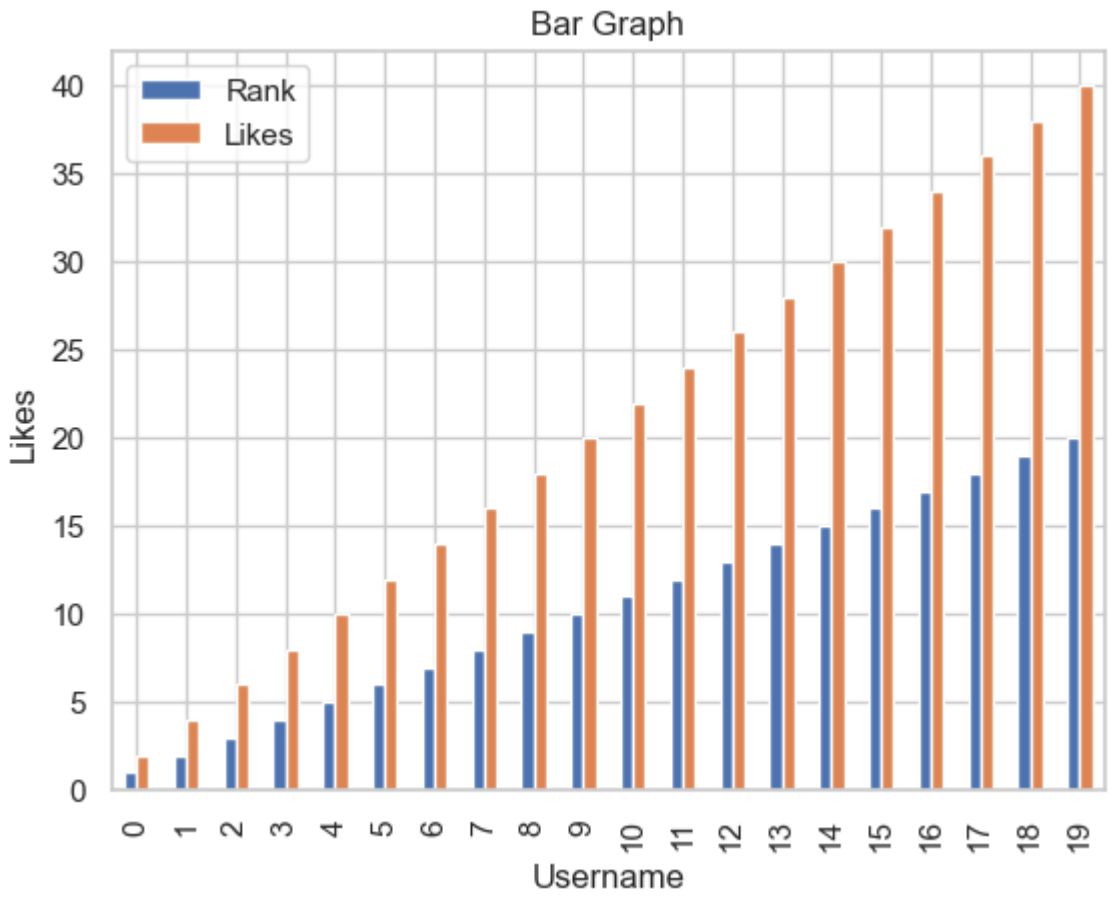
In [189]:
```
sum_result = df['Likes'].mode()
sum_result

```

Out[189]:
```
0        2
1        4
2        6
3        8
4       10
      ...
194    390
195    392
196    394
197    396
198    398
Name: Likes, Length: 199, dtype: int64
```

BAR CHART

In [190]:
```
import matplotlib.pyplot as plt
bar_data = df.head(20)
plt.figure(figsize=(8, 6))
bar_data.plot(kind='bar')
plt.xlabel('Username')
plt.ylabel('Likes')
plt.title('Bar Graph')
plt.show()
```

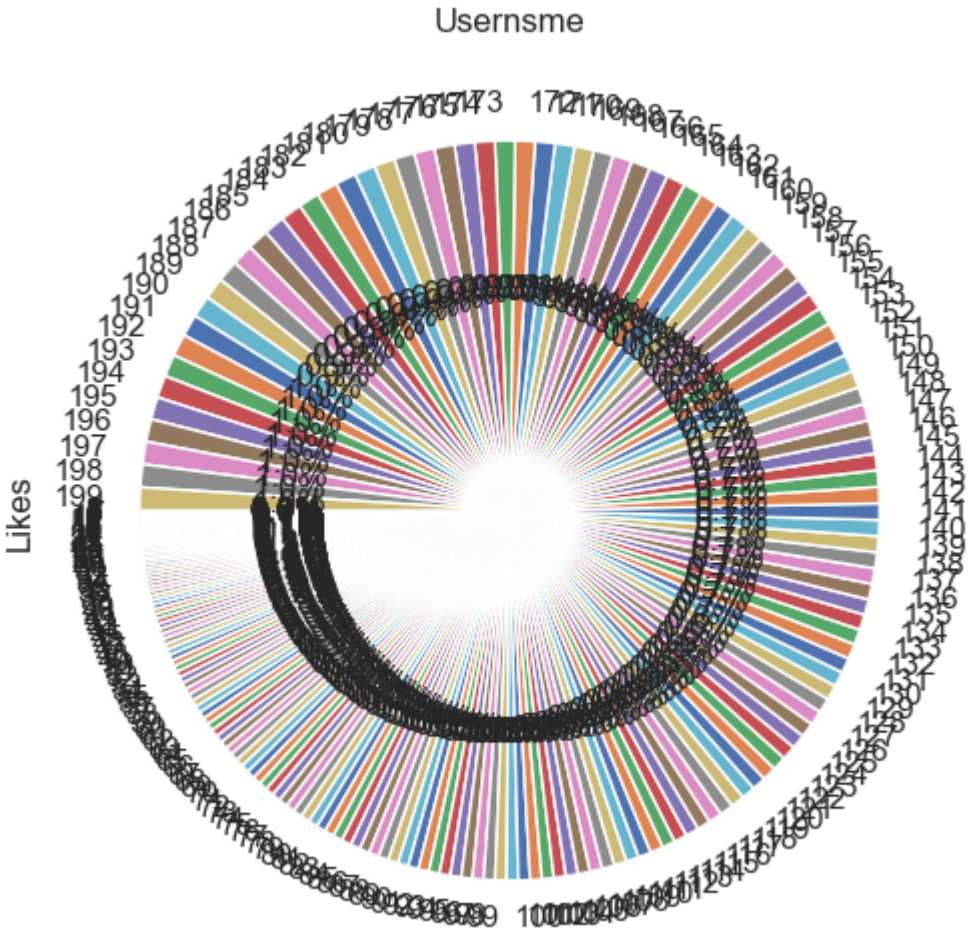
```
<Figure size 800x600 with 0 Axes>
```



PIE CHART

In [191]:

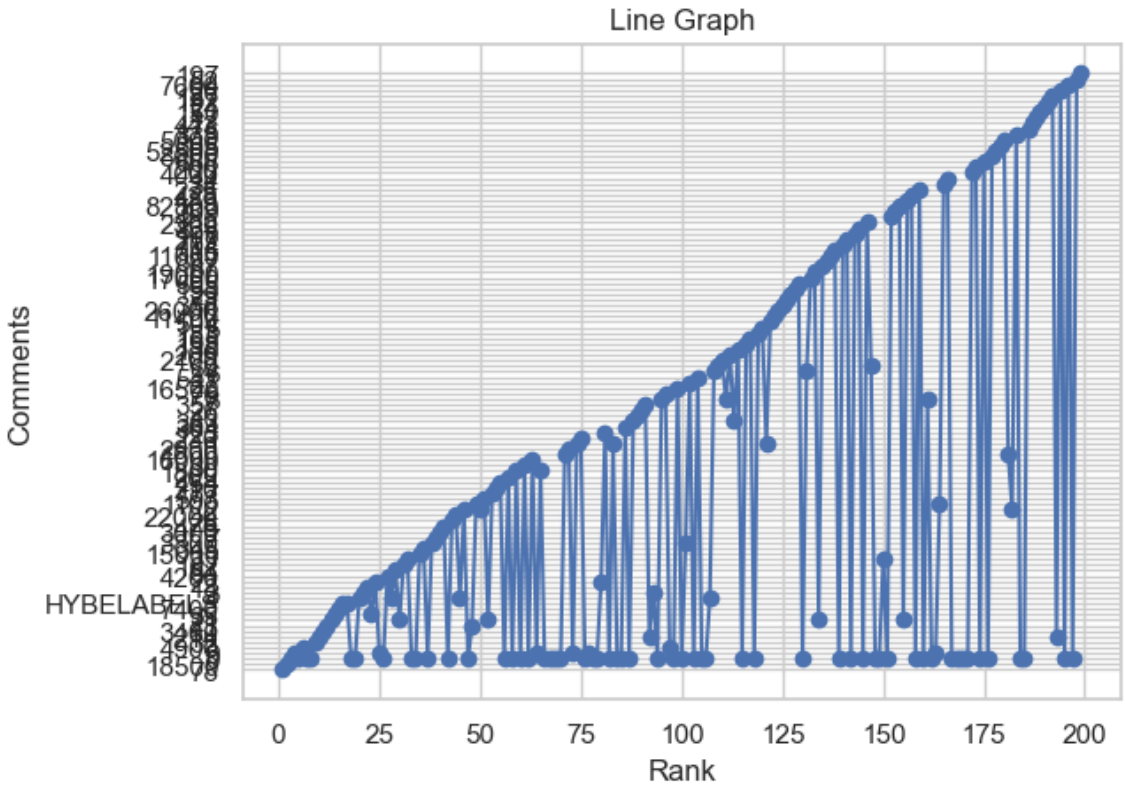
```
pie_data = df.groupby('Rank')['Likes'].sum()
plt.figure(figsize=(8, 6))
pie_data.plot(kind='pie', autopct='%1.1f%%', startangle=180)
plt.title('Usernsme')
plt.show()
```



LINE GRAPH

In [192]:

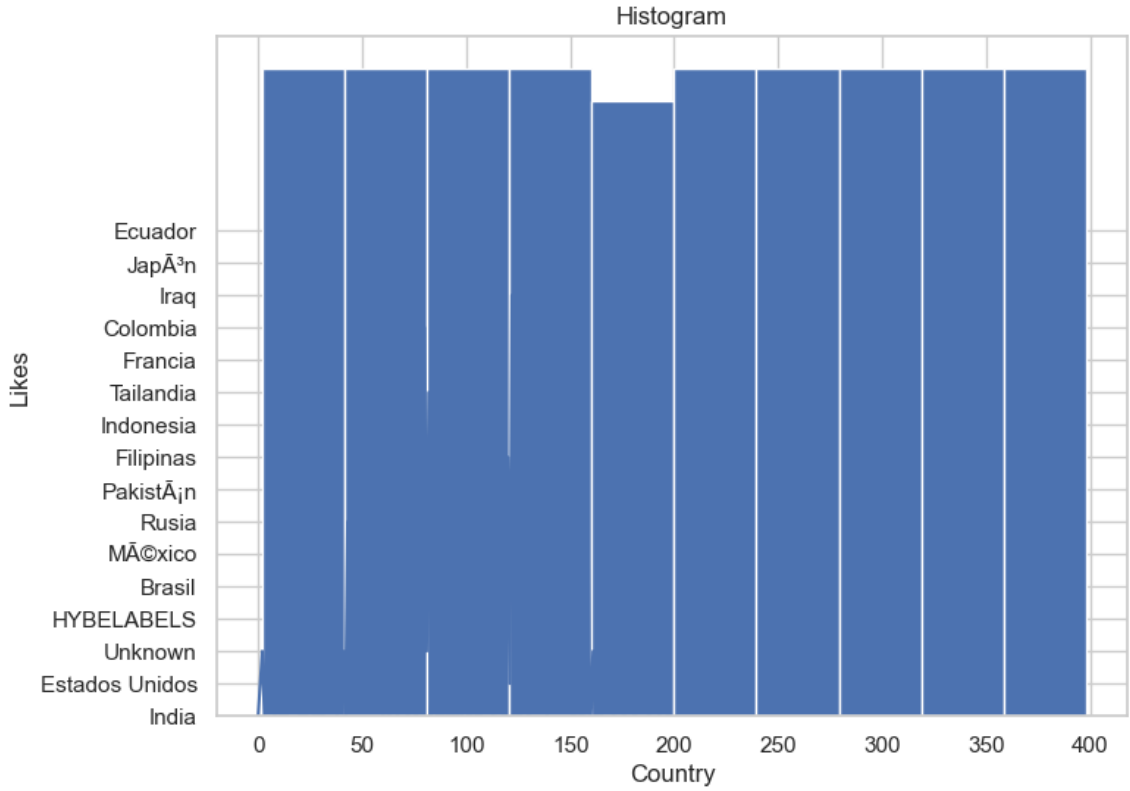
```
x_line = df['Rank']
y_line = df['Comments']
plt.plot(x_line, y_line, marker='o', linestyle='-')
plt.xlabel('Rank')
plt.ylabel('Comments')
plt.title('Line Graph')
plt.grid(True)
plt.show()
```



HISTOGRAM

In [193]:

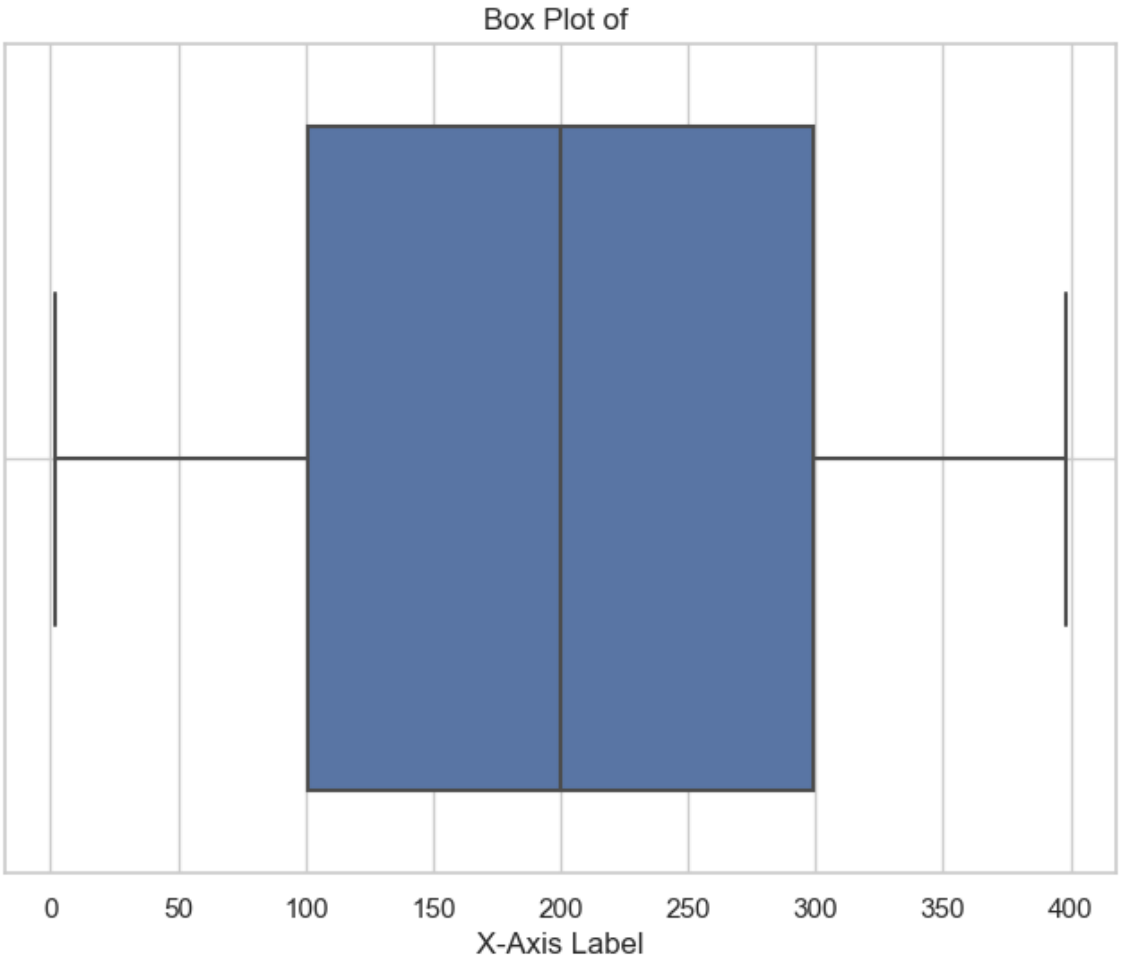
```
data_hist =df['Likes']
plt.figure(figsize=(8, 6))
plt.hist(data_hist, bins=10)
plt.xlabel('Country')
plt.ylabel('Likes')
plt.plot(df.index, df['Country'], label='Likes', color='b')

# Display the plot
plt.grid(True)  # Optional: Add grid lin')
plt.title('Histogram')
plt.show()

```



BOX PLOT

In [194]:

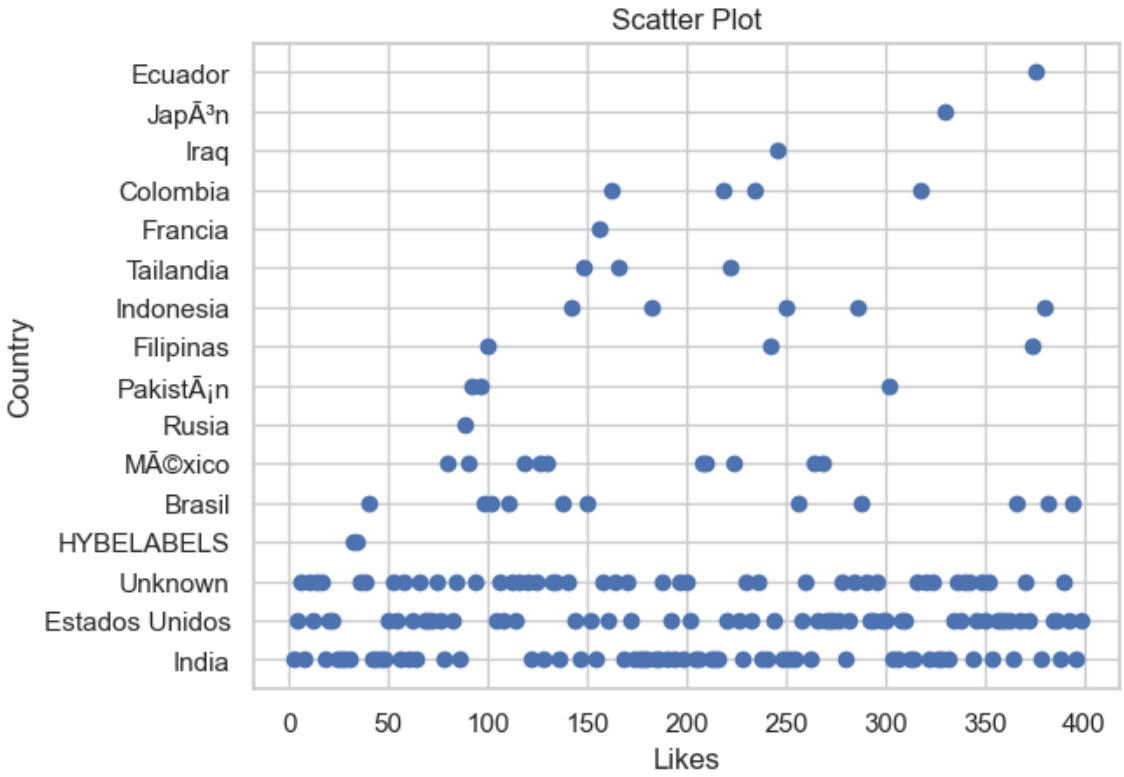
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='Likes', data=df)
plt.xlabel('X-Axis Label')
plt.title('Box Plot of ')
plt.grid(True)
plt.show()
```



SCATTER PLOT

In [195]:

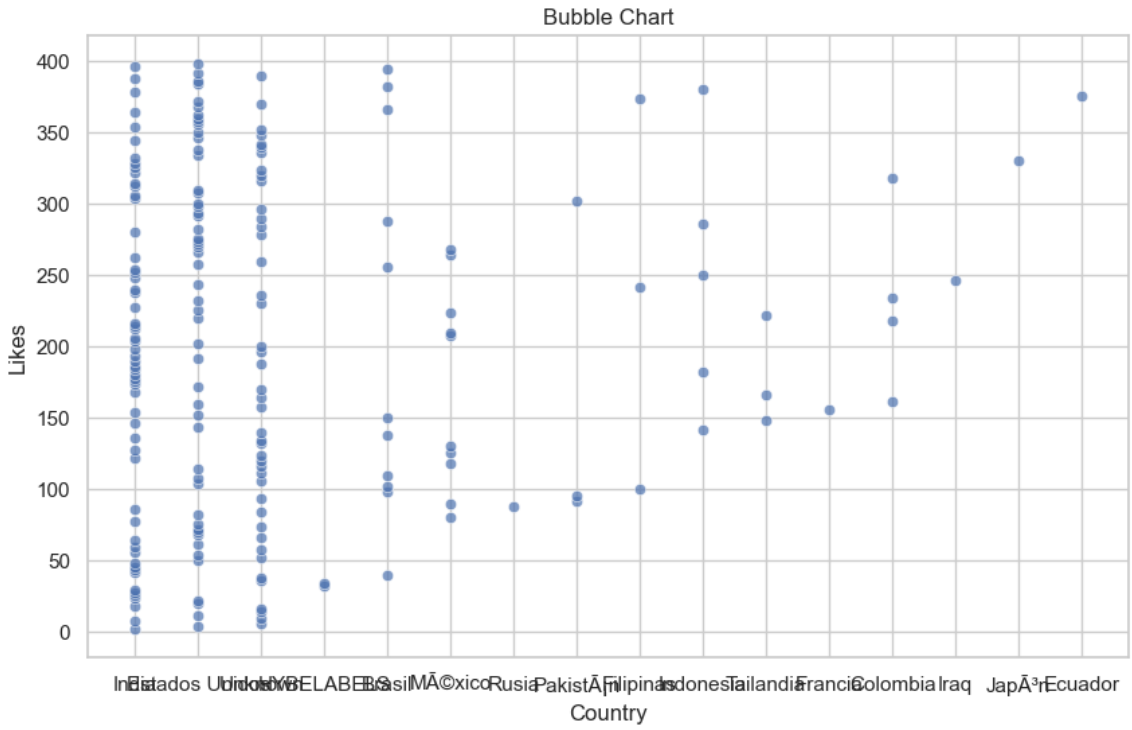
```
import pandas as pd
import matplotlib.pyplot as plt
x_column = 'Likes'
y_column = 'Country'
# Create a scatter plot
plt.scatter(df['Likes'], df['Country'])

# Add labels and title
plt.xlabel('Likes')
plt.ylabel('Country')
plt.title('Scatter Plot')

# Show the plot
plt.show()

```



BUBBLE CHART

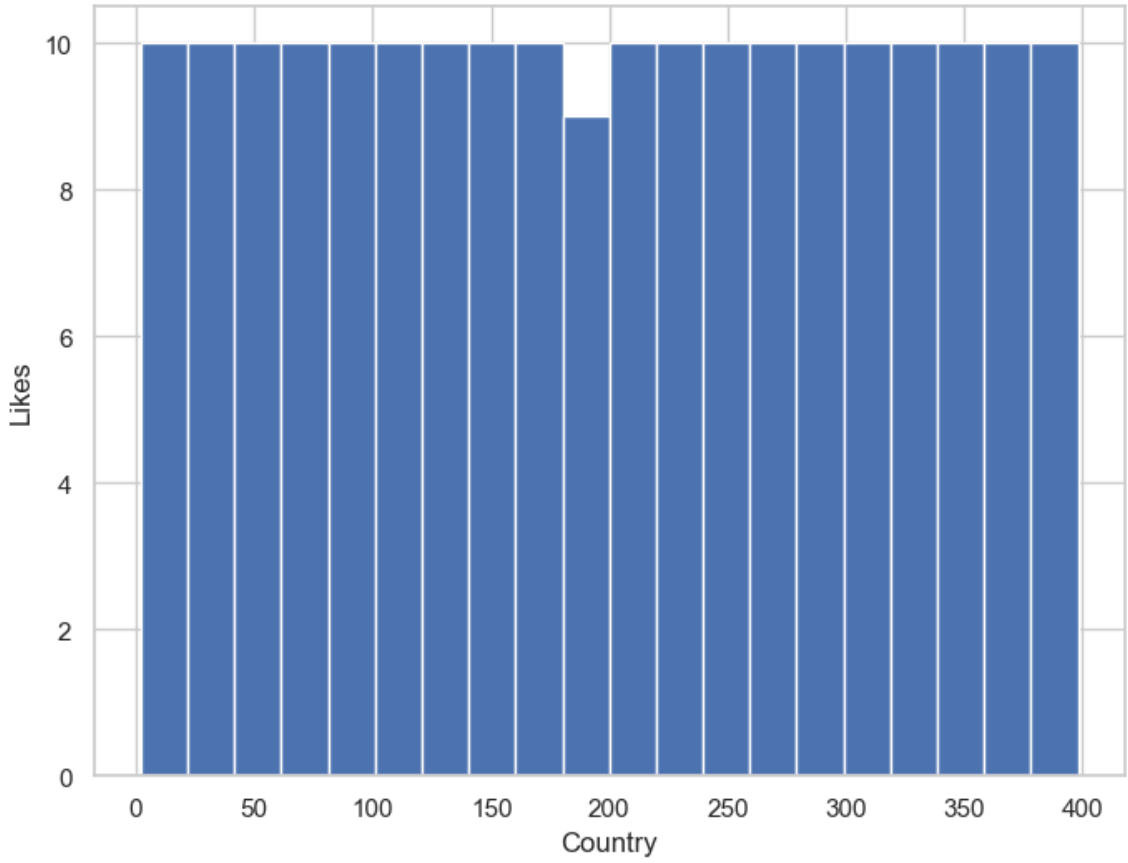
In [196]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define the columns for x, y, and bubble size
x_column = 'Country'  # Replace with the actual column name
y_column = 'Likes'  # Replace with the actual column name
size_column = 'Size_Column'  # Replace with the actual column name

# Create a bubble chart
plt.figure(figsize=(10, 6))
sns.scatterplot(x=x_column, y=y_column, data=df, sizes=(20, 200), alpha

# Customize the plot properties (title, labels, etc.)
plt.title('Bubble Chart')
plt.xlabel('Country')
plt.ylabel('Likes')

# Show the bubble chart
plt.show()
```



HISTOGRAM

In [197]:
```
passenger_age = df['Likes'].dropna()
plt.figure(figsize=(8, 6))
plt.hist(passenger_age, bins=20)
plt.xlabel("Country")
plt.ylabel("Likes")
plt.grid(True)
plt.show()
```

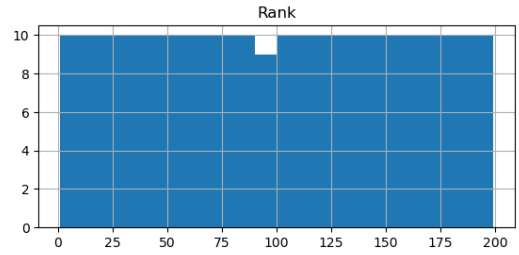


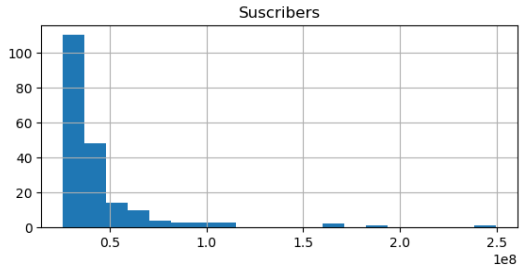
Data exploration

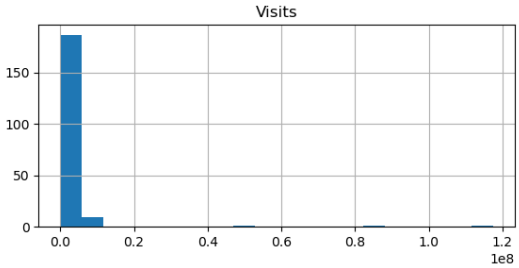
In [2]:

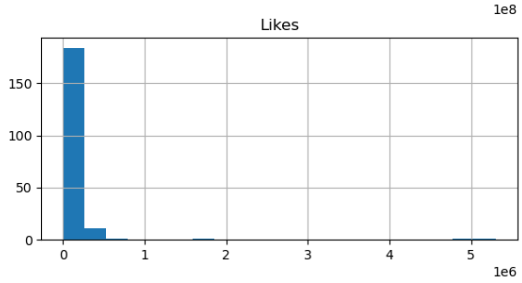
```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Check the first few rows of the dataset
print(df.head())

# Get an overview of the dataset
print(df.info())

# Generate summary statistics
print(df.describe())

# Count missing values in each column
print(df.isnull().sum())

# Visualize data: Create histograms for numeric columns
df.hist(bins=20, figsize=(15, 10))
plt.show()

# Visualize data: Create scatter plots or other visualizations as neede
# Example: data.plot(x='column_name', y='another_column', kind='scatter
# plt.show()

```

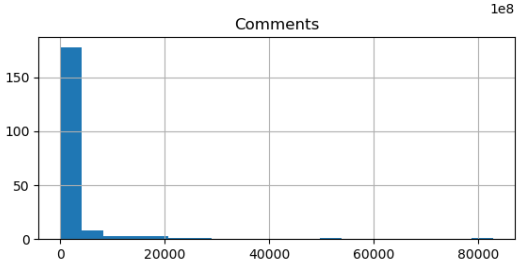
```
   Rank       Username  Categories  Suscribers         Country     Visits
\
0     1        tseries        True   249500000           India      86200
1     2        MrBeast       False   183500000  Estados Unidos  117400000
2     3      CoComelon        True   165500000         Unknown    7000000
3     4       SETIndia       False   162600000           India      15600
4     5  KidsDianaShow        True   113500000         Unknown    3900000

     Likes  Comments                                                Links
0     2700        78  http://youtube.com/channel/UCq-Fj5jknLsUf-MWSy... (h
ttp://youtube.com/channel/UCq-Fj5jknLsUf-MWSy...)
1  5300000     18500  http://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u... (h
ttp://youtube.com/channel/UCX6OQ3DkcsbYNE6H8u...)
2    24700         0  http://youtube.com/channel/UCbCmjCuTUZos6Inko4... (h
ttp://youtube.com/channel/UCbCmjCuTUZos6Inko4...)
3      166         9  http://youtube.com/channel/UCpEhnqL0y41EpW2TvW... (h
ttp://youtube.com/channel/UCpEhnqL0y41EpW2TvW...)
4    12400         0  http://youtube.com/channel/UCk8GzjMOrta8yxDcKf... (h
ttp://youtube.com/channel/UCk8GzjMOrta8yxDcKf...)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199 entries, 0 to 198
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Rank        199 non-null    int64
 1   Username    199 non-null    object
 2   Categories  199 non-null    bool
 3   Suscribers  199 non-null    int64
 4   Country     199 non-null    object
 5   Visits      199 non-null    int64
 6   Likes       199 non-null    int64
 7   Comments    199 non-null    int64
 8   Links       199 non-null    object
dtypes: bool(1), int64(5), object(3)
memory usage: 12.8+ KB
None
             Rank    Suscribers        Visits         Likes      Comments
count  199.000000  1.990000e+02  1.990000e+02  1.990000e+02    199.000000
mean   100.000000  4.405226e+07  2.261227e+06  1.047747e+05   2096.356784
std     57.590508  2.753718e+07  1.084192e+07  5.321669e+05   7913.381544
min      1.000000  2.590000e+07  0.000000e+00  0.000000e+00      0.000000
25%     50.500000  3.030000e+07  3.225000e+04  5.845000e+02      0.000000
50%    100.000000  3.550000e+07  2.257000e+05  3.900000e+03     28.000000
75%    149.500000  4.500000e+07  1.050000e+06  3.465000e+04    375.000000
max    199.000000  2.495000e+08  1.174000e+08  5.300000e+06  82800.000000
Rank          0
Username      0
Categories    0
Suscribers    0
Country       0
Visits        0
Likes         0
Comments      0
Links         0
dtype: int64
```











Correlation & Covariance

Loading [MathJax]/extensions/MathML/content-mathml.js

In [7]:
```
import pandas as pd
import numpy as np

# Calculate Correlation
correlation_matrix = df.corr()
print("Correlation Matrix:")
print(correlation_matrix)

# Calculate Covariance
covariance_matrix = df.cov()
print("\nCovariance Matrix:")
print(covariance_matrix)
```

```
Correlation Matrix:
                    Rank    Categories  Suscribers    Visits     Likes  \
Rank        1.000000e+00  4.338515e-16   -0.677636 -0.044936 -0.032294
Categories  4.338515e-16  1.000000e+00    0.017224 -0.048342 -0.020632
Suscribers -6.776364e-01  1.722411e-02    1.000000  0.255249  0.217098
Visits     -4.493650e-02 -4.834173e-02    0.255249  1.000000  0.977950
Likes      -3.229386e-02 -2.063200e-02    0.217098  0.977950  1.000000
Comments    6.149461e-02 -8.054422e-02    0.003749  0.460614  0.405251

             Comments
Rank         0.061495
Categories  -0.080544
Suscribers   0.003749
Visits       0.460614
Likes        0.405251
Comments     1.000000

Covariance Matrix:
                    Rank    Categories    Suscribers        Visits  \
Rank        3.316667e+03  5.382900e-16 -1.074650e+09 -2.805797e+07
Categories  5.382900e-16  2.512563e-01  2.377468e+05 -2.627161e+05
Suscribers -1.074650e+09  2.377468e+05  7.582961e+14  7.620610e+13
Visits     -2.805797e+07 -2.627161e+05  7.620610e+13  1.175472e+14
Likes      -9.897345e+05 -5.503610e+03  3.181439e+12  5.642487e+12
Comments    2.802529e+04 -3.194883e+02  8.168526e+08  3.951894e+10

                   Likes      Comments
Rank       -9.897345e+05  2.802529e+04
Categories -5.503610e+03 -3.194883e+02
Suscribers  3.181439e+12  8.168526e+08
Visits      5.642487e+12  3.951894e+10
Likes       2.832016e+11  1.706609e+09
Comments    1.706609e+09  6.262161e+07
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

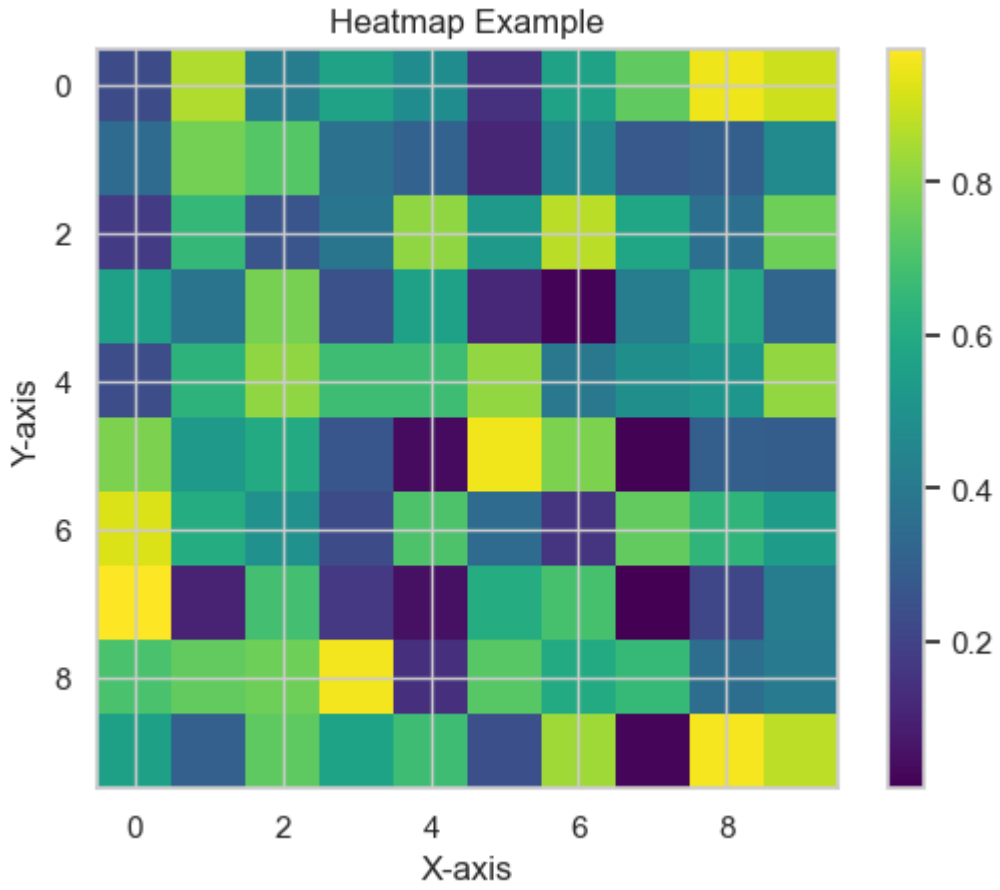
Loading [MathJax]/extensions/MathML/content-mathml.js

In [ ]:
```
1
```

In [ ]:
```
1
```

HEATMAP

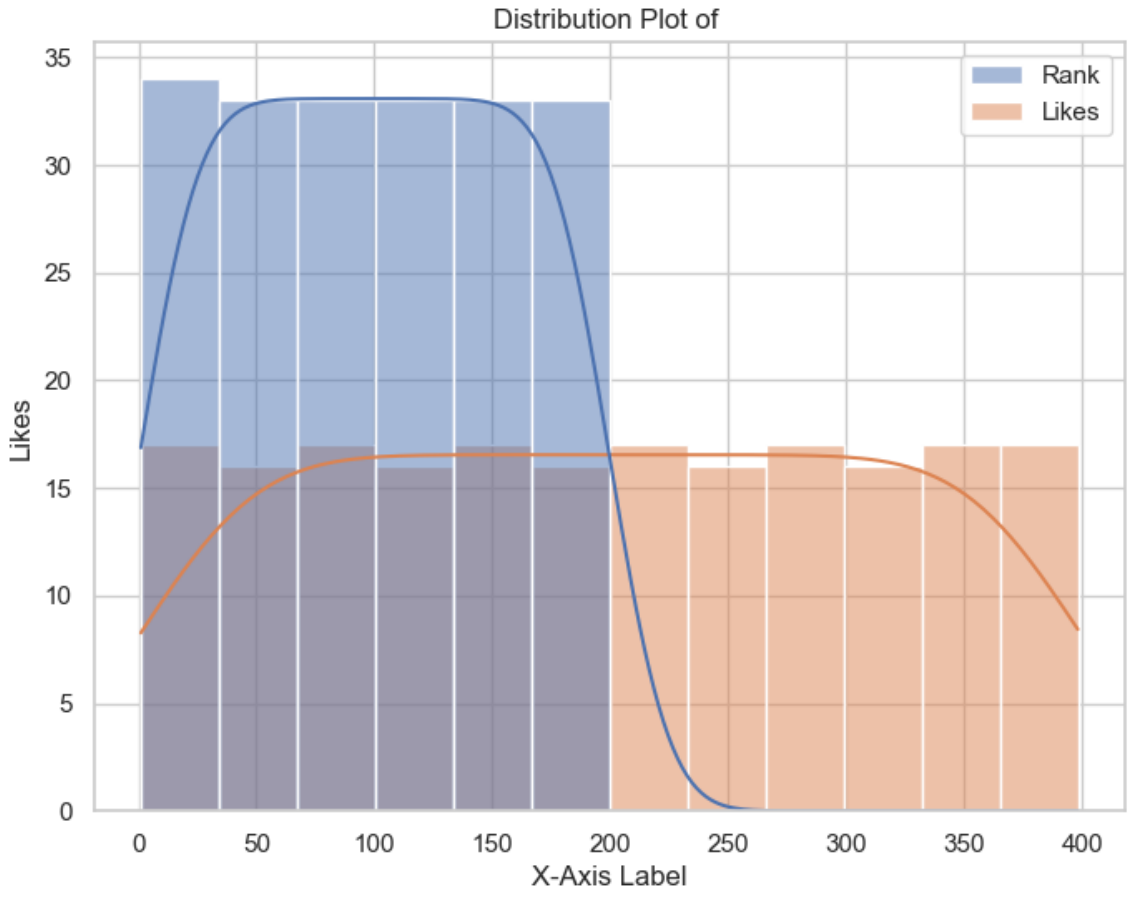
In [229]:
```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # Sample data for the heatmap
5  data = np.random.random((10, 10))  # Replace this with your own data
6
7  # Create a heatmap
8  plt.imshow(data, cmap='viridis')  # You can choose a different colormap
9  plt.colorbar()  # Add a colorbar for reference
10
11 plt.title('Heatmap Example')
12 plt.xlabel('X-axis')
13 plt.ylabel('Y-axis')
14
15 # Show the heatmap
16 plt.show()
```



DISTREBUTION PLOT

In [199]:

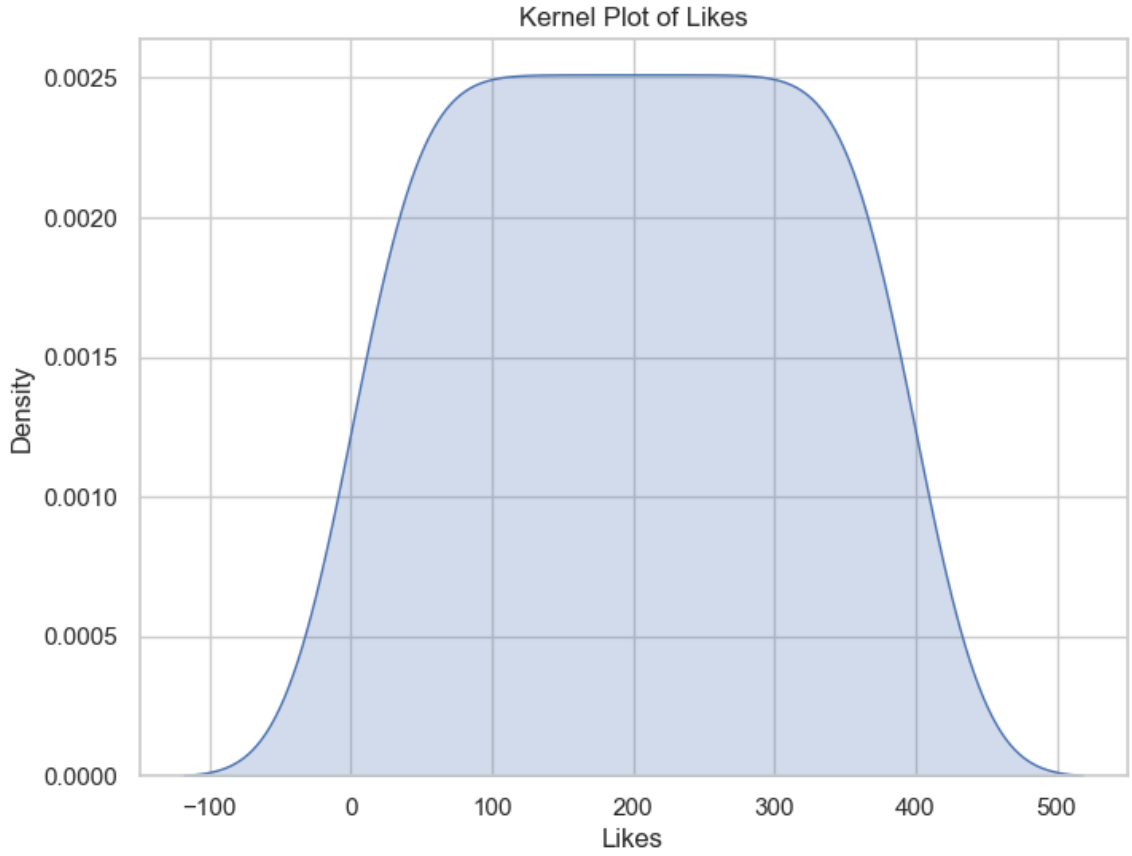
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Extract the variable for which you want to create a distribution plot
variable_to_plot = 'Rank'  # Replace 'ColumnName' with the actual colum

# Create a distribution plot using Seaborn
plt.figure(figsize=(8, 6))  # Optional: Set the figure size
sns.histplot(data=df, kde=True)  # 'kde=True' adds a kernel density est

# Customize the plot (labels, title, etc.)
plt.xlabel('X-Axis Label')
plt.ylabel('Likes')
plt.title('Distribution Plot of ')

# Show the distribution plot
plt.show()
```



KERNAL PLOT

In [200]:

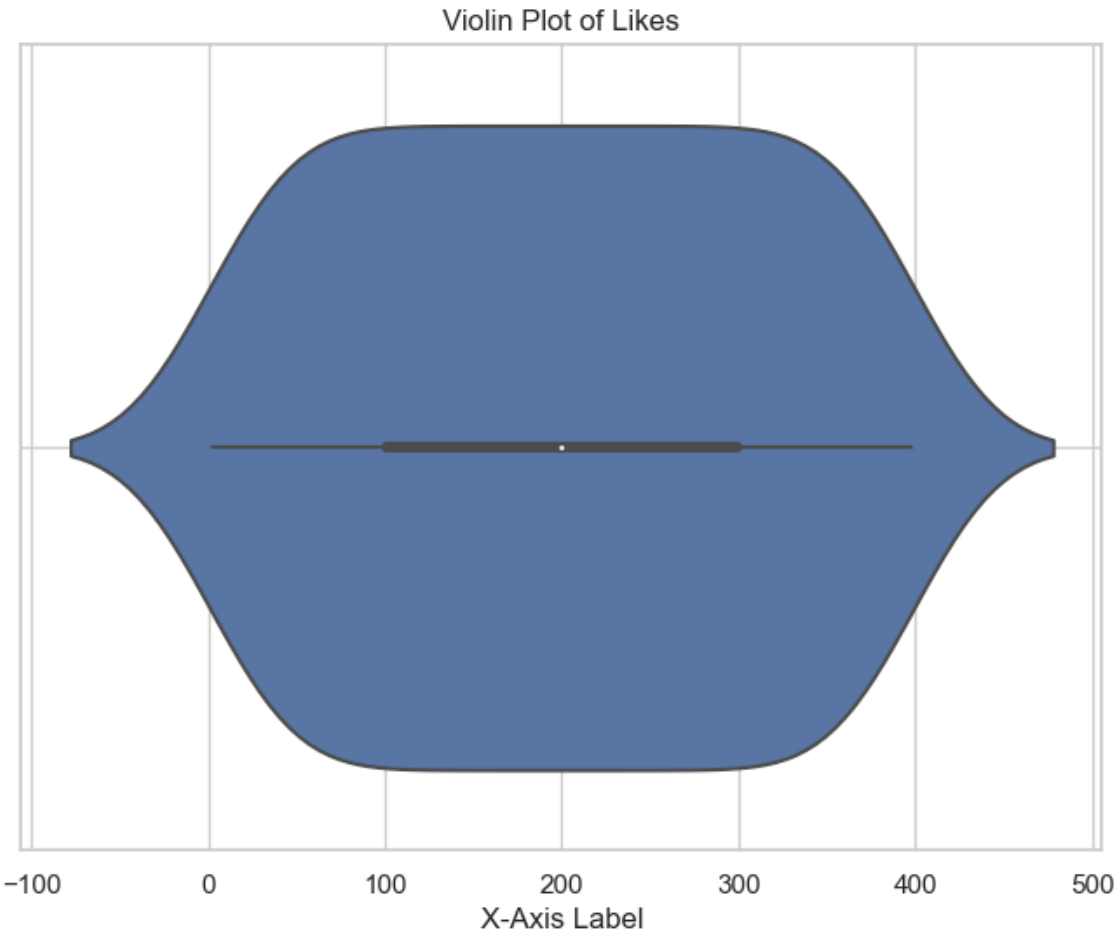
```
import seaborn as sns
import matplotlib.pyplot as plt

# Extract the variable for which you want to create a KDE plot
variable_to_plot = 'Likes'  # Replace 'ColumnName' with the actual colu

# Create a KDE plot
plt.figure(figsize=(8, 6))
sns.kdeplot(df['Likes'], shade=True)
plt.xlabel('Likes')
plt.ylabel('Density')
plt.title('Kernel Plot of ' + variable_to_plot)
plt.grid(True)
plt.show()
```



VIOLIN PLOT

In [201]:

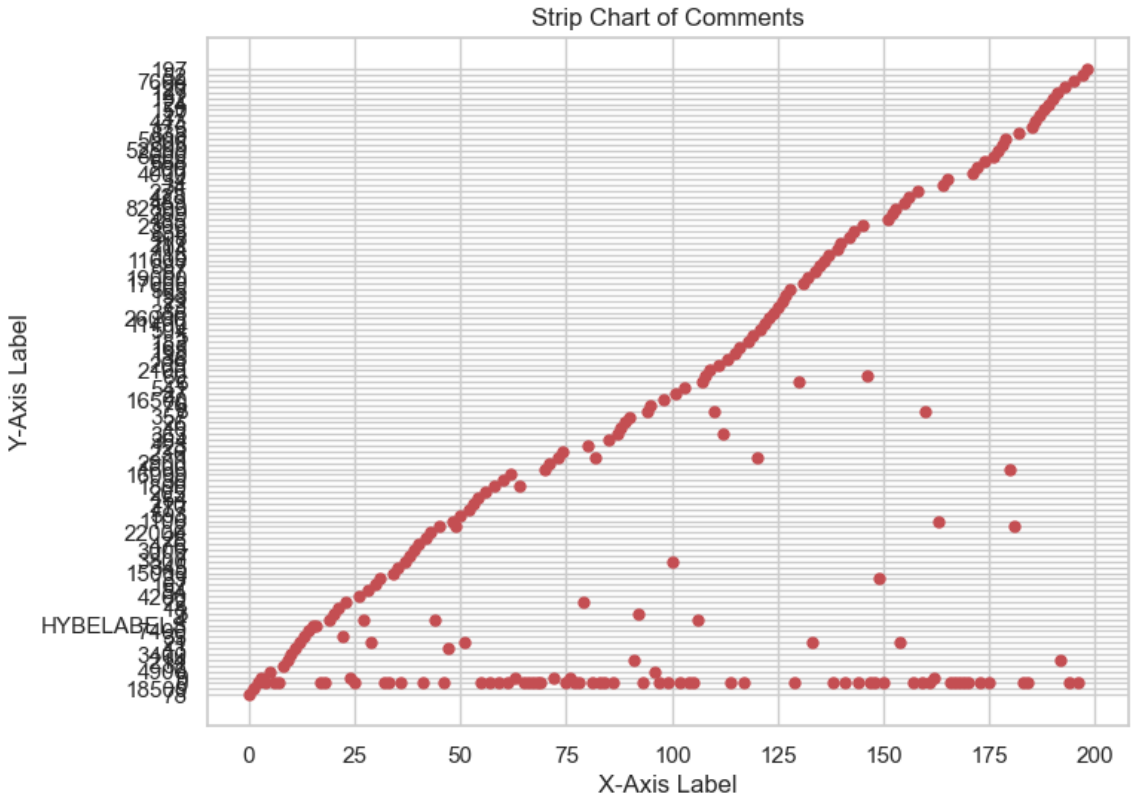
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a violin plot
plt.figure(figsize=(8, 6))
sns.violinplot(x='Likes', data=df)
plt.xlabel('X-Axis Label')
plt.title('Violin Plot of ' + variable_to_plot)
plt.grid(True)
plt.show()
```



STRIP CHART

In [202]:

```
 import pandas as pd
import matplotlib.pyplot as plt
# Extract the variable for the strip chart
variable_to_plot = 'Comments'  # Replace 'ColumnName' with the actual c

# Create a strip chart
plt.figure(figsize=(8, 6))
plt.plot(df[variable_to_plot], 'ro', markersize=5)  # 'ro' means red ci
plt.xlabel('X-Axis Label')
plt.ylabel('Y-Axis Label')
plt.title('Strip Chart of ' + variable_to_plot)
plt.grid(True)
plt.show()
```

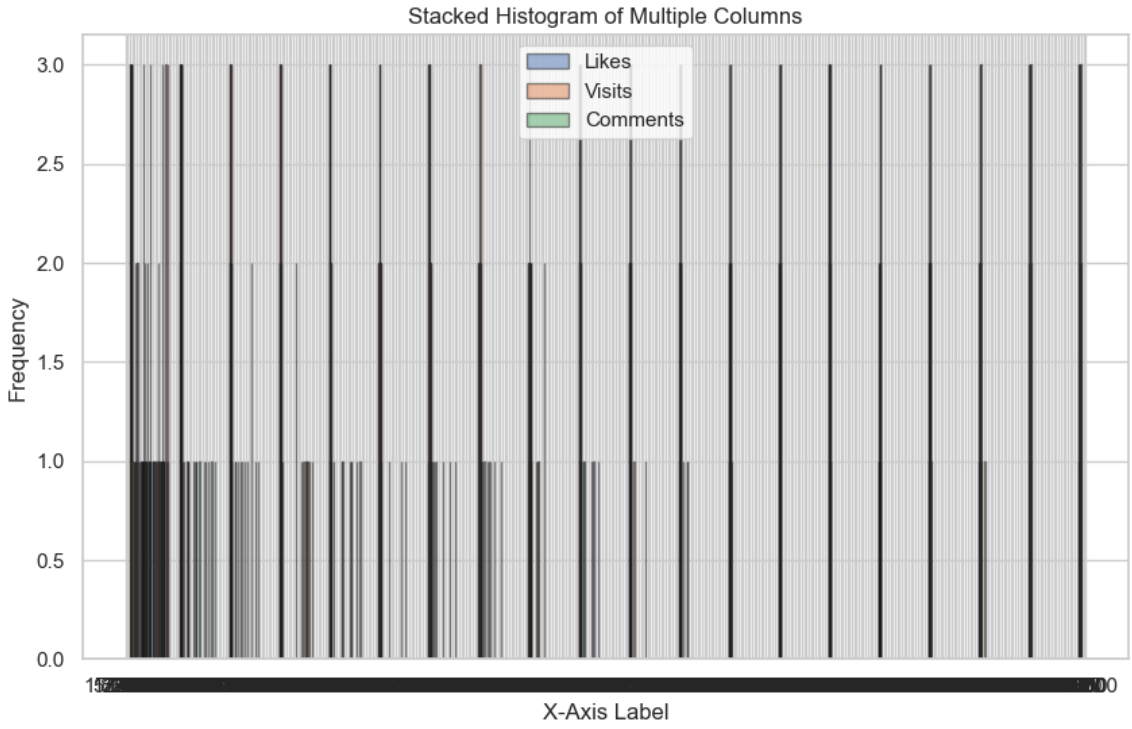


Loading [MathJax]/extensions/MathML/content-mathml.js

In [230]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Define the columns you want to create a stacked histogram for
columns_to_plot = ['Likes', 'Visits', 'Comments']  # Replace with your

# Create a stacked histogram
plt.figure(figsize=(10, 6))
plt.hist(df[columns_to_plot].values.T, bins=20, edgecolor='k', alpha=0.
plt.xlabel('X-Axis Label')
plt.ylabel('Frequency')
plt.title('Stacked Histogram of Multiple Columns')
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\IPython\core\pylabtools.py:1
51: UserWarning: Creating legend with loc="best" can be slow with large am
ounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```

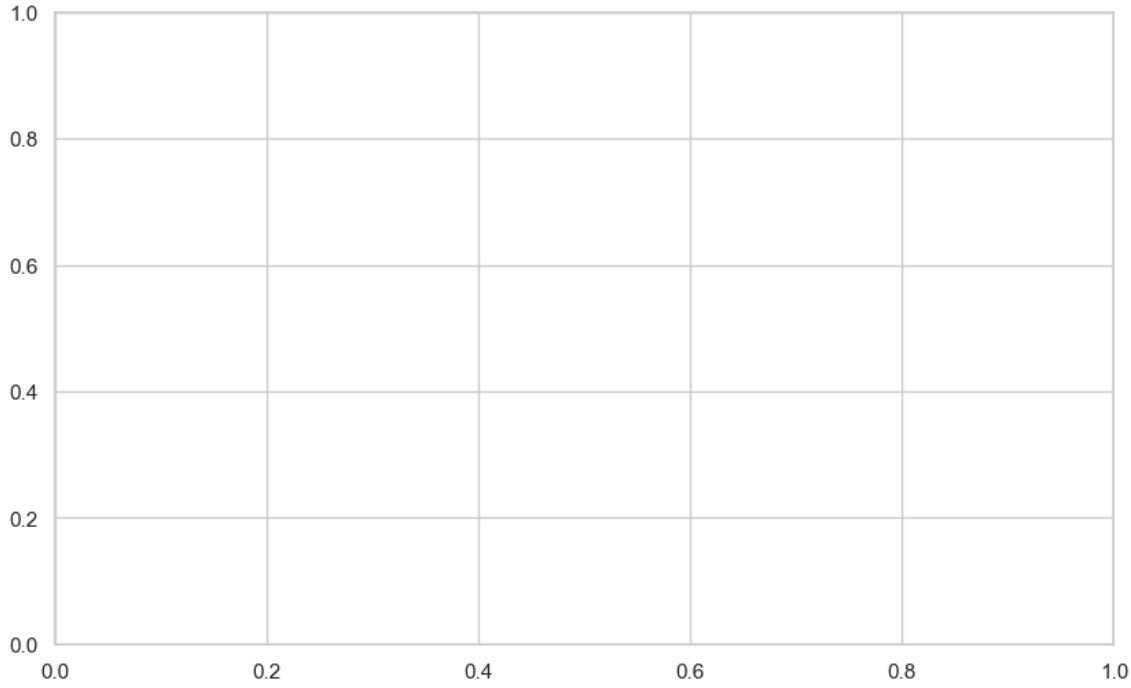


Overlapping Density Plots

In [231]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Define the columns you want to create overlapping density plots for
columns_to_plot = ['Likes', 'Rank']  # Replace with your column names

# Create overlapping density plots
plt.figure(figsize=(10, 6))
for column in columns_to_plot:
    sns.kdeplot(df[column], label=column, shade=True)

plt.xlabel('X-Axis Label')
plt.ylabel('Density')
plt.title('Overlapping Density Plots of Multiple Columns')
plt.legend()
plt.grid(True)
plt.show()
```

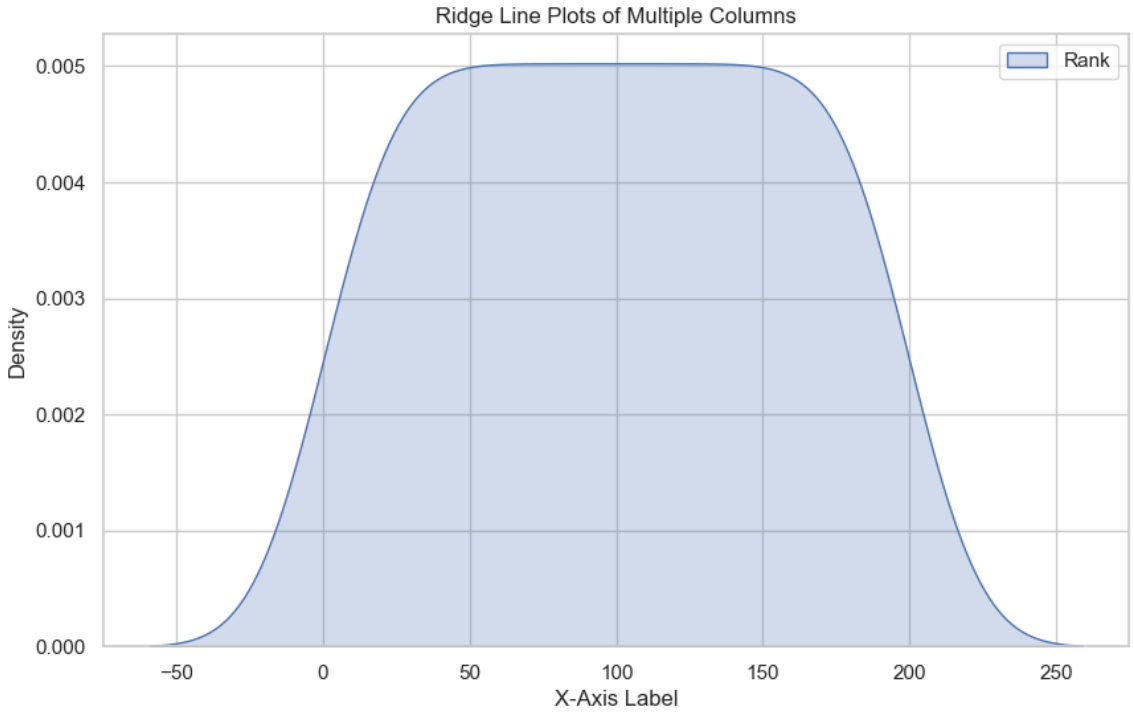
```
---------------------------------------------------------------------------
-
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_12064\2074536078.py in <module>
      8 plt.figure(figsize=(10, 6))
      9 for column in columns_to_plot:
---> 10     sns.kdeplot(df[column], label=column, shade=True)
     11
     12 plt.xlabel('X-Axis Label')

~\anaconda3\lib\site-packages\seaborn\_decorators.py in inner_f(*args, **k
wargs)
     44             )
     45         kwargs.update({k: arg for k, arg in zip(sig.parameters, ar
gs)})
---> 46         return f(**kwargs)
     47     return inner_f
     48

~\anaconda3\lib\site-packages\seaborn\distributions.py in kdeplot(x, y, sh
ade, vertical, kernel, bw, gridsize, cut, clip, legend, cumulative, shade_
lowest, cbar, cbar_ax, cbar_kws, ax, weights, hue, palette, hue_order, hue
_norm, multiple, common_norm, common_grid, levels, thresh, bw_method, bw_a
djust, log_scale, color, fill, data, data2, warn_singular, **kwargs)
   1760     )
   1761
-> 1762     p._attach(ax, allowed_types=["numeric", "datetime"], log_scale
=log_scale)
   1763
   1764     if p.univariate:

~\anaconda3\lib\site-packages\seaborn\_core.py in _attach(self, obj, allow
ed_types, log_scale)
   1123                     f"{allowed_types} is required"
   1124                 )
-> 1125                 raise TypeError(err)
   1126
   1127             # Register with the matplotlib unit conversion machine
ry

TypeError: The x variable is categorical, but one of ['numeric', 'datetim
e'] is required
```



Ridge Line Plots

In [235]:

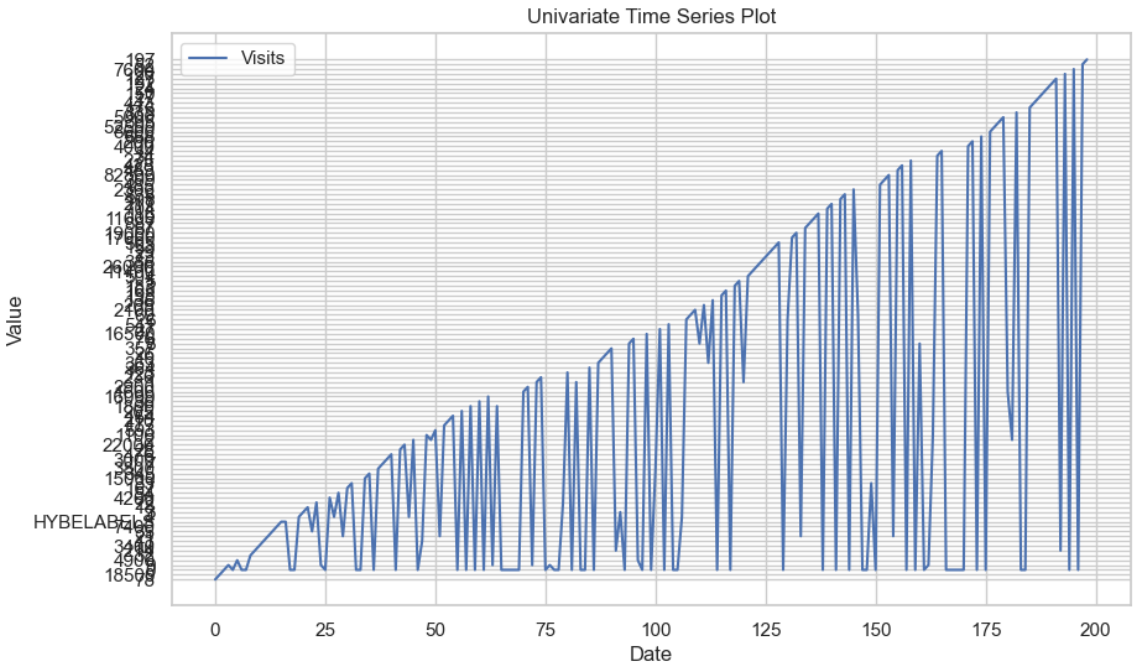
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Define the columns you want to create ridge line plots for
columns_to_plot = ['Rank']  # Replace with your column names

# Create ridge line plots
plt.figure(figsize=(10, 6))
sns.set(style='whitegrid')
for column in columns_to_plot:
    sns.kdeplot(df[column], label=column, shade=True)

plt.xlabel('X-Axis Label')
plt.ylabel('Density')
plt.title('Ridge Line Plots of Multiple Columns')
plt.legend()
plt.grid(True)
plt.show()
```



Univariate Time Series Plot

In [232]:

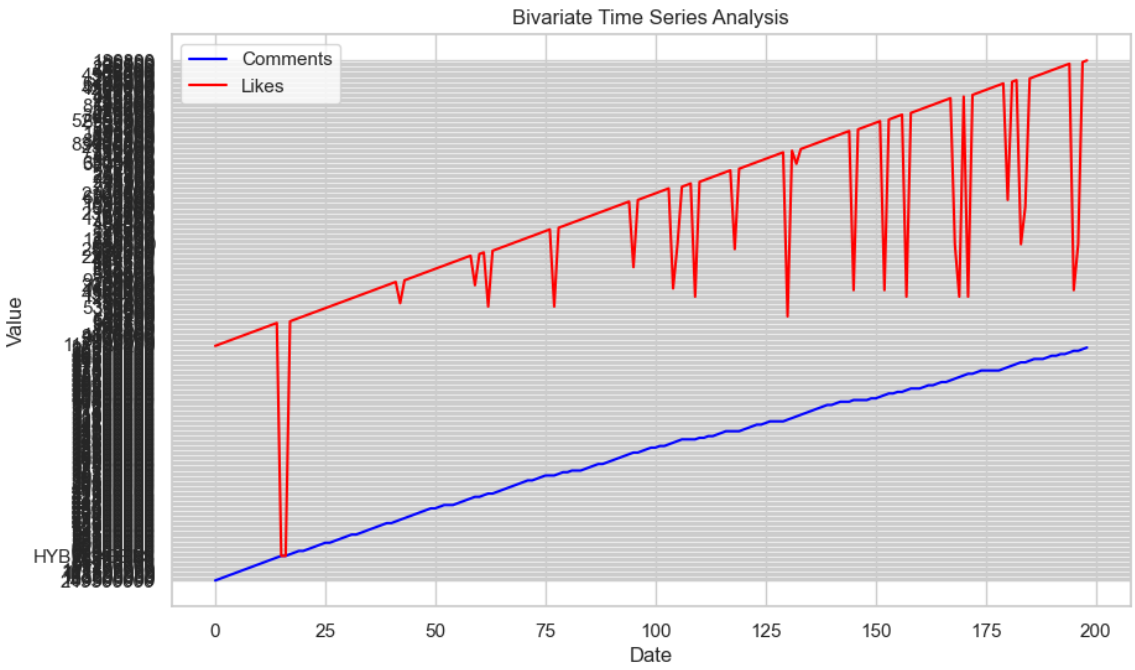
```
plt.figure(figsize=(10, 6))  # Optional: Set the figure size

# Plot the time series data
plt.plot(df.index, df['Comments'], label='Visits', color='b')

# Optional: Add labels and a title
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Univariate Time Series Plot')

# Optional: Customize other plot properties

# Display the plot
plt.grid(True)  # Optional: Add grid lines
plt.legend()  # Optional: Show the legend if you have multiple series
plt.show()
```



Bivariate Time Series Analysis

In [233]:

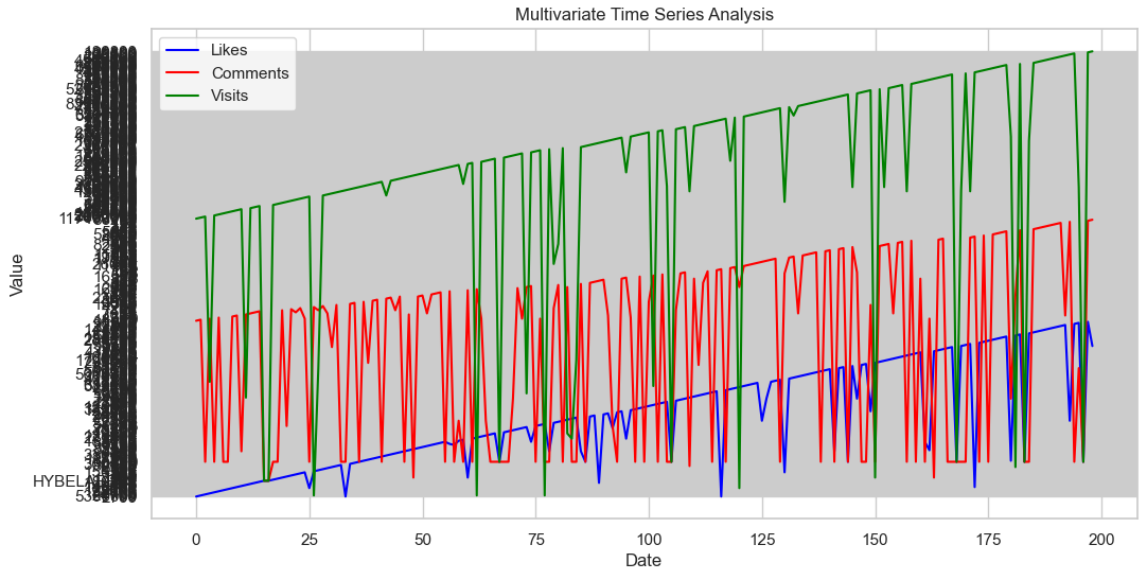
```
import pandas as pd
import matplotlib.pyplot as plt


# Plot the time series data
plt.figure(figsize=(10, 6))
plt.plot(df.index, df['Suscribers'], label='Comments', color='blue')
plt.plot(df.index, df['Visits'], label='Likes', color='red')

# Customize the plot
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Bivariate Time Series Analysis')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



Multivariate Time Series Analysis

In [243]:

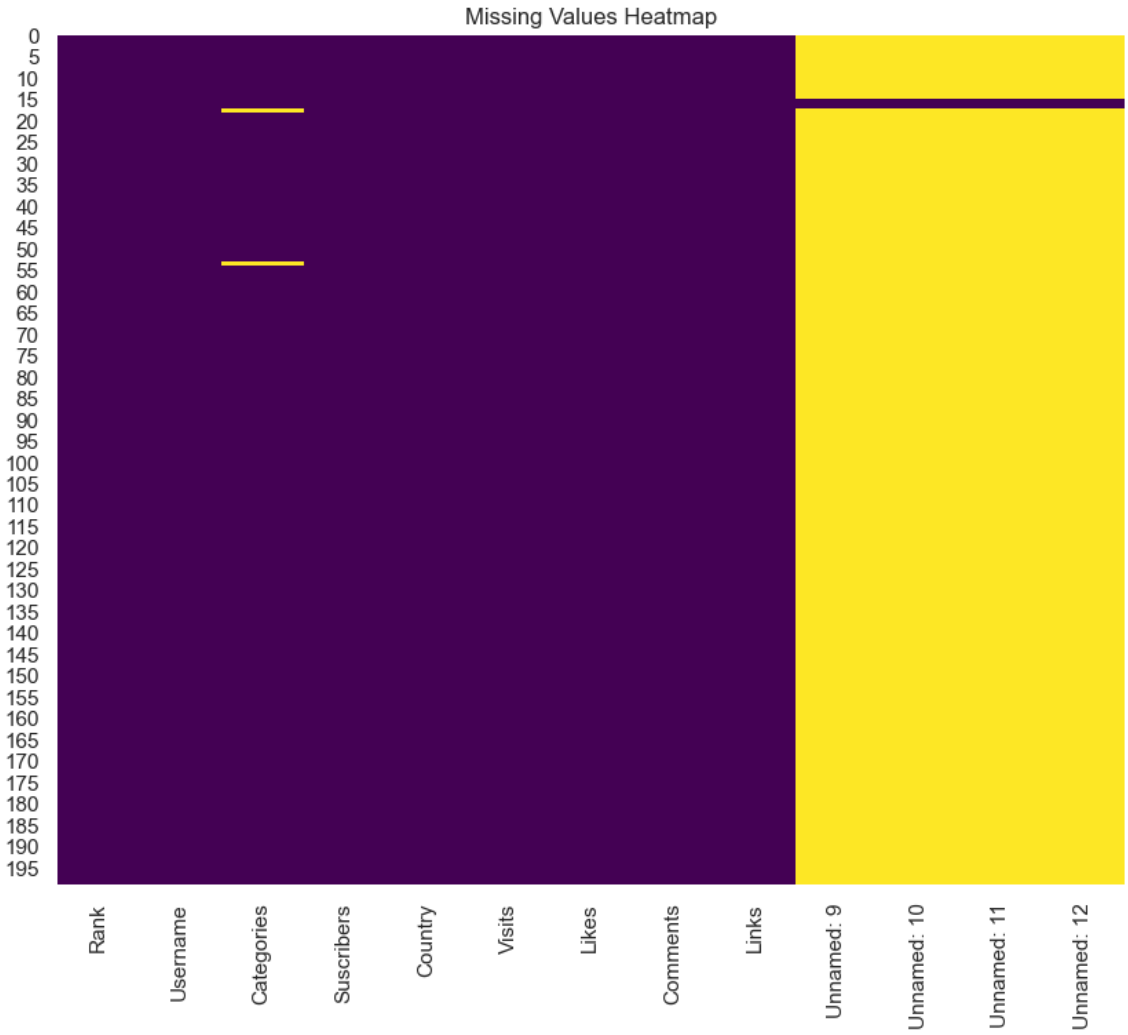
```
import pandas as pd
import matplotlib.pyplot as plt


# Plot the multivariate time series data
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['Likes'], label='Likes', color='blue')
plt.plot(df.index, df['Comments'], label='Comments', color='red')
plt.plot(df.index, df['Visits'], label='Visits', color='green')

# Customize the plot
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Multivariate Time Series Analysis')
plt.legend()
plt.grid(True)

# Show the plot
plt.show()
```



Missing Values Heatmap

In [237]:

```
 import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Load your CSV file
file_path = r"C:\Users\Anusha V\Desktop\YouTube.csv"
df = pd.read_csv(file_path)
# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Values Heatmap')
plt.show()
```



isnull method

In [239]:
```
1 df.isnull()
```

Out[239]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comments	Links	Unr
0	False	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	...	
194	False	False	False	False	False	False	False	False	False	
195	False	False	False	False	False	False	False	False	False	
196	False	False	False	False	False	False	False	False	False	
197	False	False	False	False	False	False	False	False	False	
198	False	False	False	False	False	False	False	False	False	

199 rows × 13 columns

Number of missing values

In [240]:
```
1 df.isnull().sum().sum()
```

Out[240]:
```
790
```

In [241]:
```
1 df.isnull().sum()
```

Out[241]:
```
Rank             0
Username         0
Categories       2
Suscribers       0
Country          0
Visits           0
Likes            0
Comments         0
Links            0
Unnamed: 9     197
Unnamed: 10    197
Unnamed: 11    197
Unnamed: 12    197
dtype: int64
```

In [242]:

```
 import pandas as pd


# Identify missing values
missing_data = df.isnull()

# Count missing values per column
missing_count = missing_data.sum()

# Count missing values per row
missing_count_per_row = missing_data.sum(axis=1)

# Print or analyze the missing data
print("Missing Data Count per Column:")
print(missing_count)

print("Missing Data Count per Row:")
print(missing_count_per_row)
```

```
Missing Data Count per Column:
Rank             0
Username         0
Categories       2
Suscribers       0
Country          0
Visits           0
Likes            0
Comments         0
Links            0
Unnamed: 9     197
Unnamed: 10    197
Unnamed: 11    197
Unnamed: 12    197
dtype: int64
Missing Data Count per Row:
0      4
1      4
2      4
3      4
4      4
      ..
194    4
195    4
196    4
197    4
198    4
Length: 199, dtype: int64
```

In [244]:
```
import pandas as pd


# Check for missing values
missing_values = df.isnull().sum()

# Print the columns with missing values
print("Columns with missing values:")
print(missing_values[missing_values > 0])
```

```
Columns with missing values:
Categories       2
Unnamed: 9     197
Unnamed: 10    197
Unnamed: 11    197
Unnamed: 12    197
dtype: int64
```

fillna method

In [218]:
```
d_f = df.fillna(100)
d_f
```

Out[218]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Con
0	1	tseries	MÃºsica y baile	249500000	India	86200	2700	
1	2	MrBeast	Videojuegos, Humor	183500000	Estados Unidos	117400000	5300000	
2	3	CoComelon	EducaciÃ³n	165500000	Unknown	7000000	24700	
3	4	SETIndia	EducaciÃ³n	162600000	India	15600	166	
4	5	KidsDianaShow	AnimaciÃ³n, Juguetes	113500000	Unknown	3900000	12400	
...	...	...	...	...	...	...	...	
194	195	dianaandromahin8102	AnimaciÃ³n, Juguetes	26200000	Unknown	109000	322	
195	196	nickiminaj	MÃºsica y baile	26100000	Estados Unidos	1600000	98300	
196	197	mariliamendoncareal	MÃºsica y baile	26100000	Brasil	0	0	
197	198	TheLallantop	Noticias y PolÃtica	26000000	India	30800	822	
198	199	AGT	MÃºsica y baile, PelÃ-culas	25900000	Estados Unidos	186800	3100	

199 rows × 13 columns

In [245]:
```
df_filled = df.fillna(1212)
df_filled
```

Out[245]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comı
0	1	tseries	4567.0	249500000	India	86200	2700	
1	2	MrBeast	4567.0	183500000	Estados Unidos	117400000	5300000	
2	3	CoComelon	4567.0	165500000	Unknown	7000000	24700	
3	4	SETIndia	4567.0	162600000	India	15600	166	
4	5	KidsDianaShow	4567.0	113500000	Unknown	3900000	12400	
...	...	...	...	...	...	...	...	
194	195	dianaandromahin8102	4567.0	26200000	Unknown	109000	322	
195	196	nickiminaj	4567.0	26100000	Estados Unidos	1600000	98300	
196	197	mariliamendoncareal	4567.0	26100000	Brasil	0	0	
197	198	TheLallantop	4567.0	26000000	India	30800	822	
198	199	AGT	4567.0	25900000	Estados Unidos	186800	3100	

199 rows × 13 columns

dropna method

In [247]:
```
d_f = df.dropna(0)
d_f
```

```
C:\Users\Anusha V\AppData\Local\Temp\ipykernel_12064\1675708418.py:1: Futu
reWarning: In a future version of pandas all arguments of DataFrame.dropna
will be keyword-only.
  d_f = df.dropna(0)
```

Out[247]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes
15	16	HYBELABELS	4567.0	HYBELABELS	HYBELABELS	HYBELABELS	HYBELABELS
16	17	HYBELABELS	4567.0	HYBELABELS	HYBELABELS	HYBELABELS	HYBELABELS

Replace

In [248]:
```
import numpy as np
df12 = df.replace(to_replace=np.nan, value='not found')
df12
```

Out[248]:

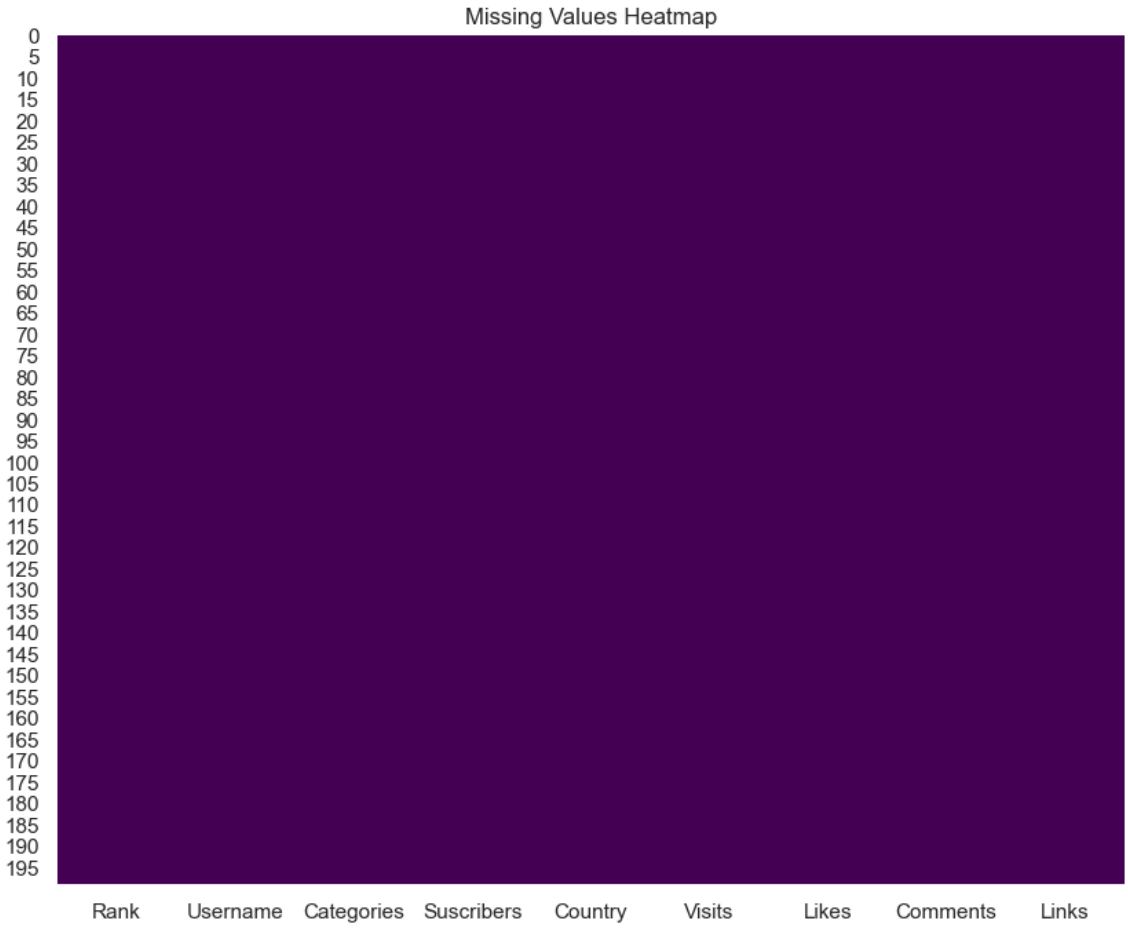
	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comi
0	1	tseries	4567.0	249500000	India	86200	2700	
1	2	MrBeast	4567.0	183500000	Estados Unidos	117400000	5300000	
2	3	CoComelon	4567.0	165500000	Unknown	7000000	24700	
3	4	SETIndia	4567.0	162600000	India	15600	166	
4	5	KidsDianaShow	4567.0	113500000	Unknown	3900000	12400	
...	...	...	...	...	...	...	...	
194	195	dianaandromahin8102	4567.0	26200000	Unknown	109000	322	
195	196	nickiminaj	4567.0	26100000	Estados Unidos	1600000	98300	
196	197	mariliamendoncareal	4567.0	26100000	Brasil	0	0	
197	198	TheLallantop	4567.0	26000000	India	30800	822	
198	199	AGT	4567.0	25900000	Estados Unidos	186800	3100	

199 rows × 13 columns

HEATMAP

In [251]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
file_path = r"C:\Users\Anusha V\Desktop\YouTube.csv"
df1 = pd.read_csv(file_path)
# Fill missing values
df1['Likes'].fillna(df1['Likes'].mean(), inplace=True)
df1['Comments'].fillna(df1['Comments'].mean(), inplace=True)
df1['Visits'].fillna(df1['Visits'].mean(), inplace=True)


# Heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df1.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Values Heatmap')
plt.show()
```



Loading [MathJax]/extensions/MathML/content-mathml.js

In [291]:

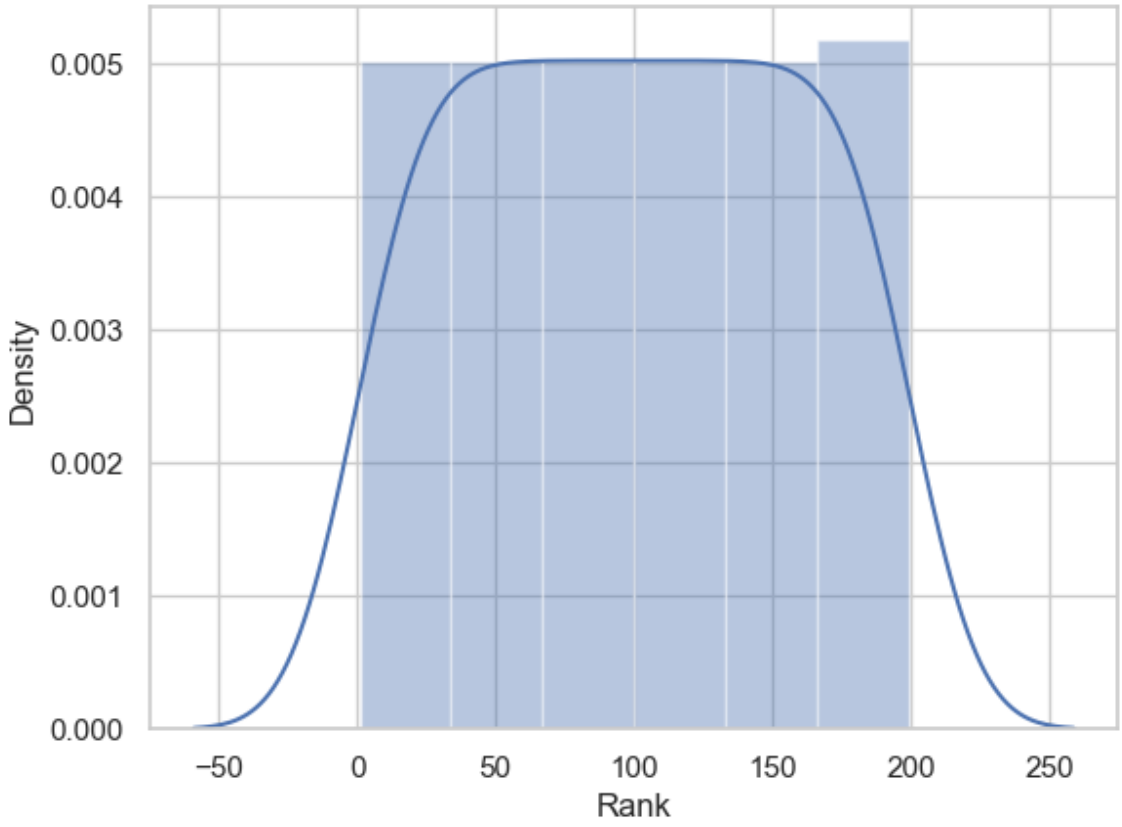
```
 import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Load your CSV file
file_path = r"C:\Users\Anusha V\Desktop\YouTube.csv"
df = pd.read_csv(file_path)
# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Values Heatmap')
plt.show()
```

Distplot

In [276]:
```
1 import seaborn as sns
2 sns.distplot(df['Rank'])
```

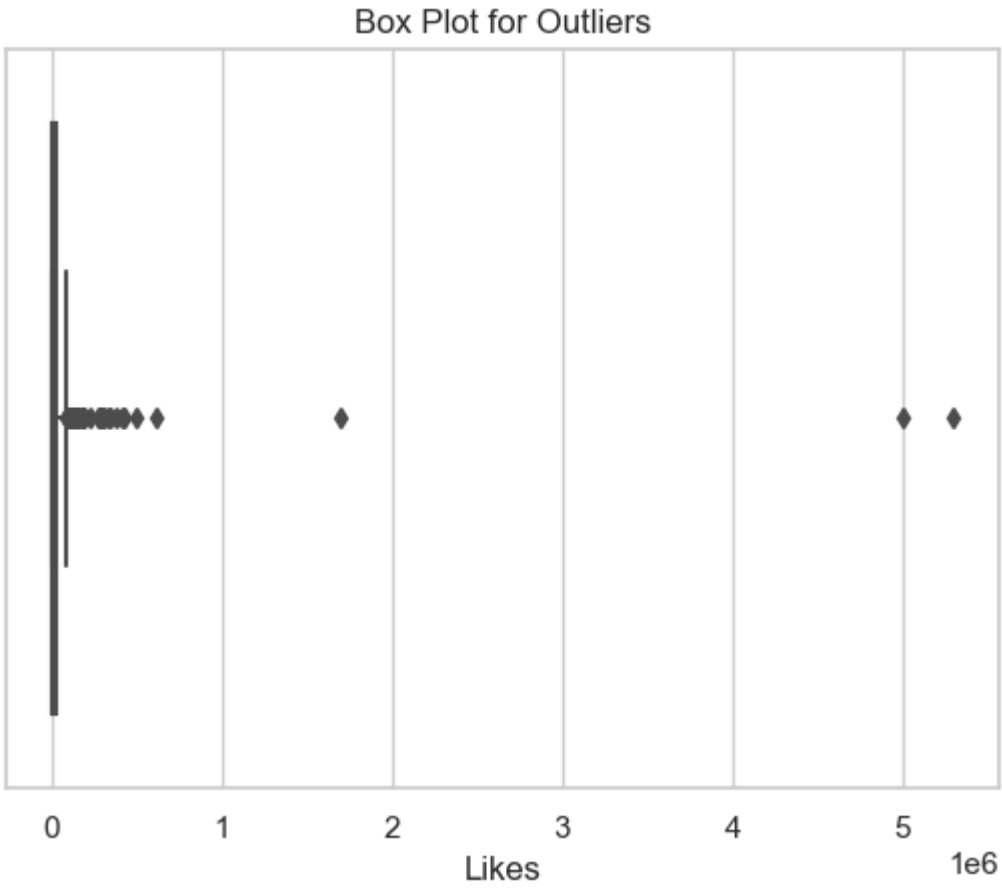
```
C:\Users\Anusha V\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[276]: <AxesSubplot:xlabel='Rank', ylabel='Density'>



Box Plot for Outliers

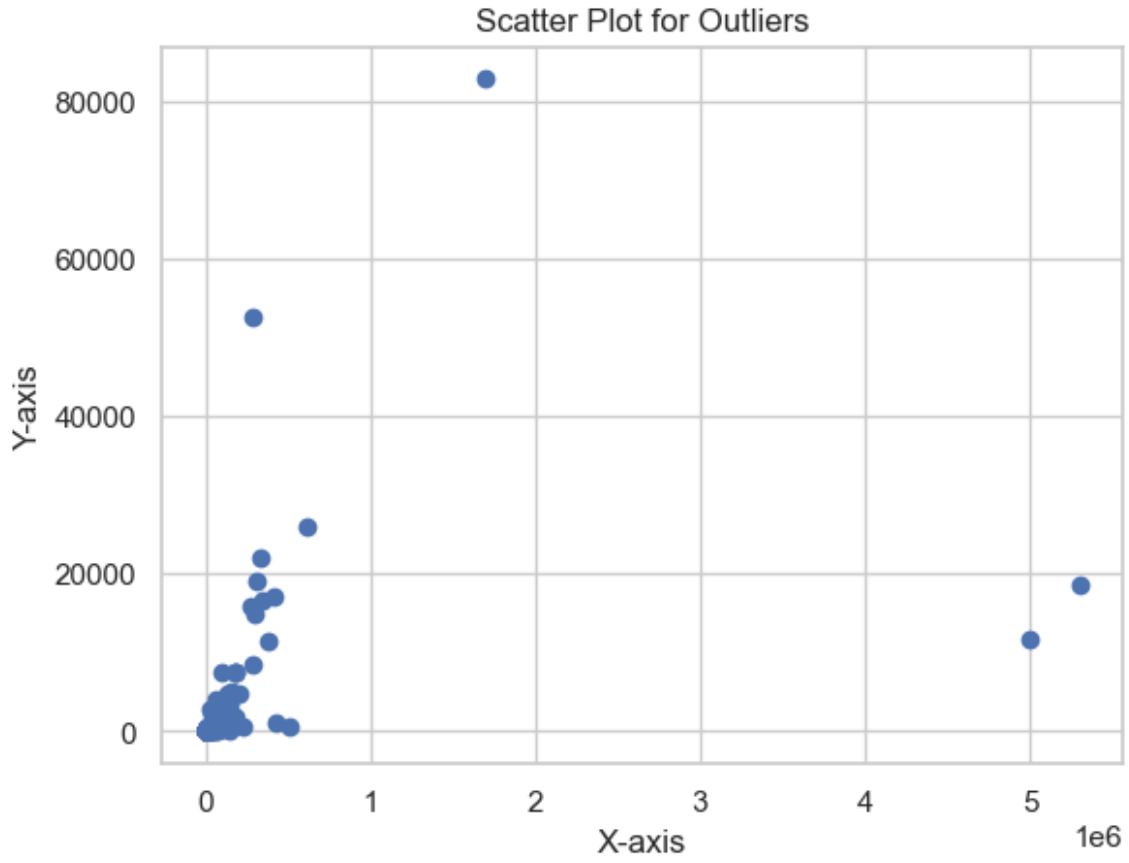
In [256]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame
sns.boxplot(x=df['Likes'])
plt.title("Box Plot for Outliers")
plt.show()
```



Scatter Plot for Outliers

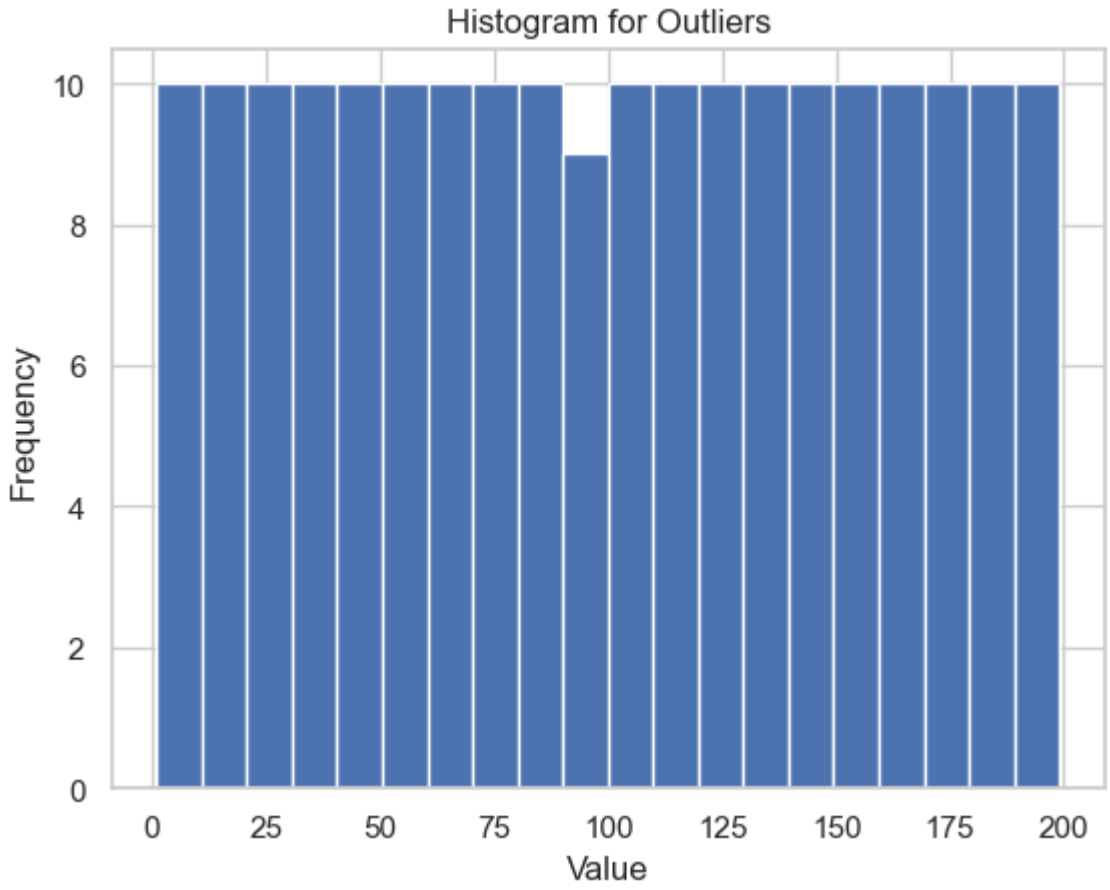
In [257]:

```
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and 'x' and 'y' are columns
plt.scatter(df['Likes'], df['Comments'])
plt.title("Scatter Plot for Outliers")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()

```



Histogram for Outliers

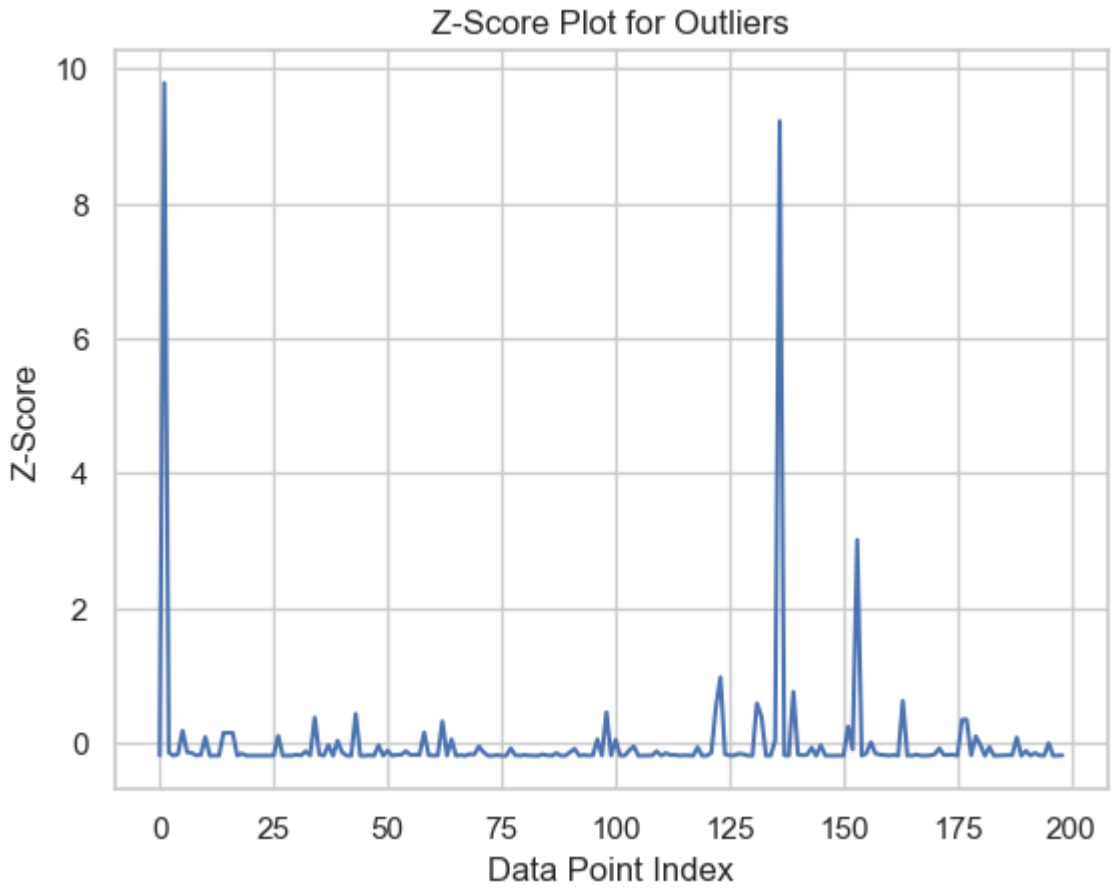
In [260]:
```
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and 'column_name' is the variable of
plt.hist(df['Rank'], bins=20)  # Adjust the number of bins as needed
plt.title("Histogram for Outliers")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

```



Z-Score Plot for Outliers

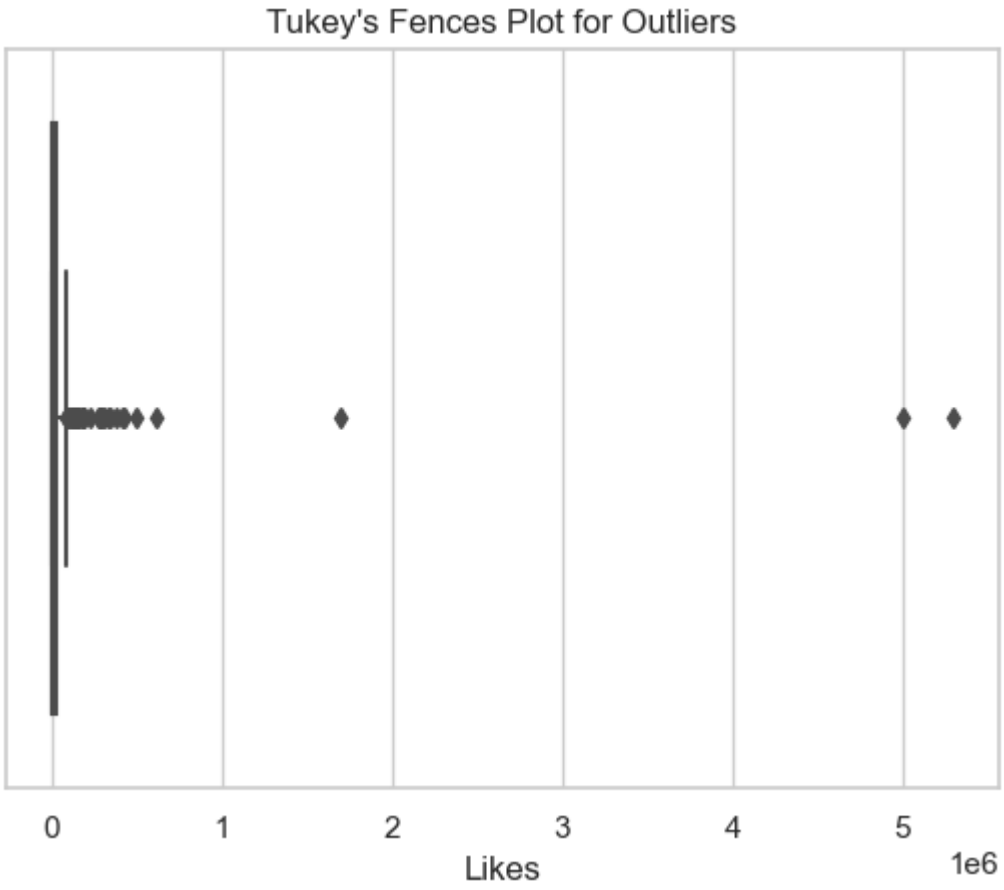
In [261]:

```
from scipy import stats

# Assuming 'df' is your DataFrame and 'column_name' is the variable of
z_scores = stats.zscore(df['Likes'])
plt.plot(z_scores)
plt.title("Z-Score Plot for Outliers")
plt.xlabel("Data Point Index")
plt.ylabel("Z-Score")
plt.show()

```



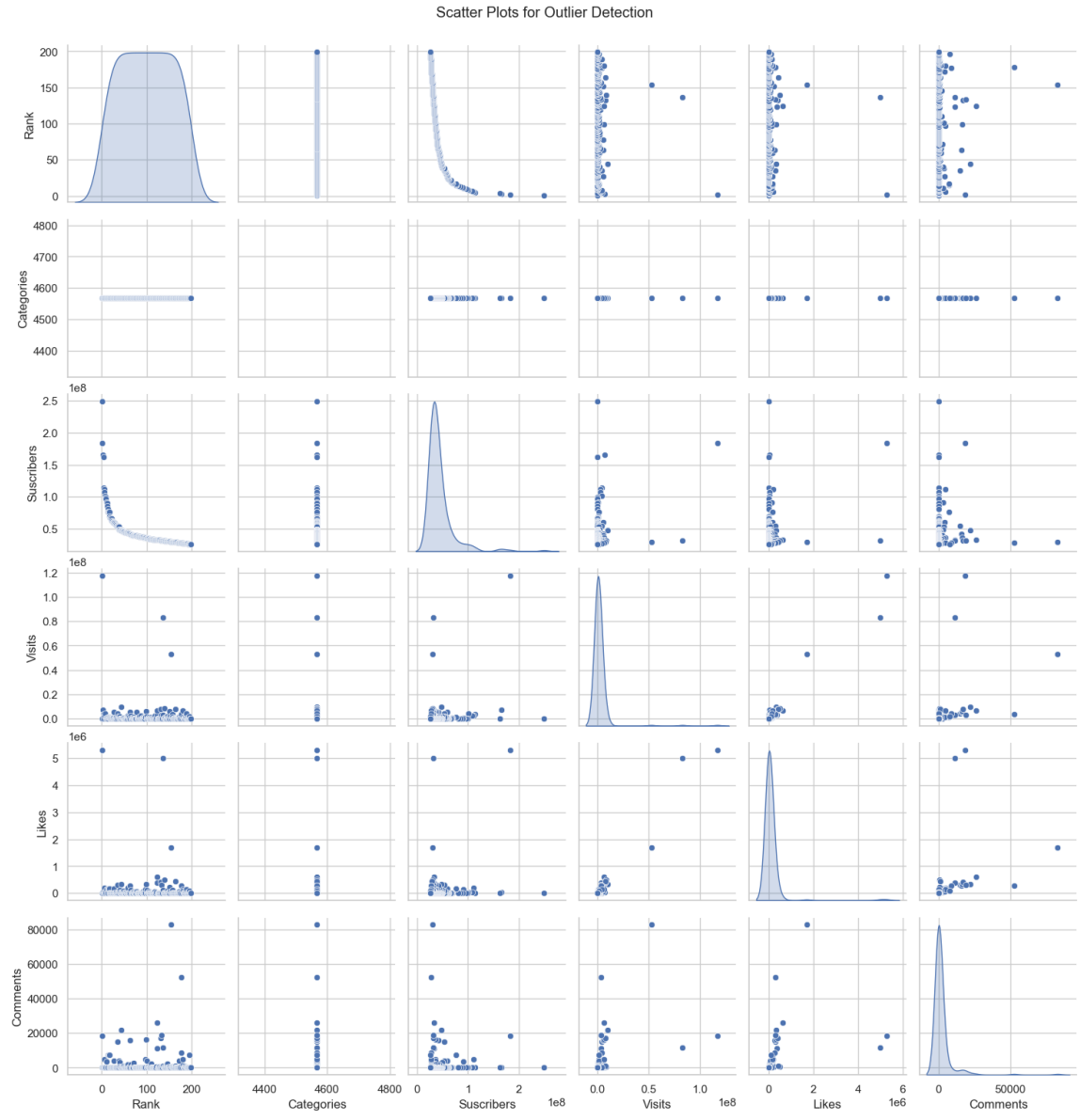
Tukey's Fences Plot for Outliers

In [263]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame and 'column_name' is the variable of
sns.boxplot(x=df['Likes'], whis=1.5)  # Adjust the whisker length as ne
plt.title("Tukey's Fences Plot for Outliers")
plt.show()

```



Scatter Plots for Outlier

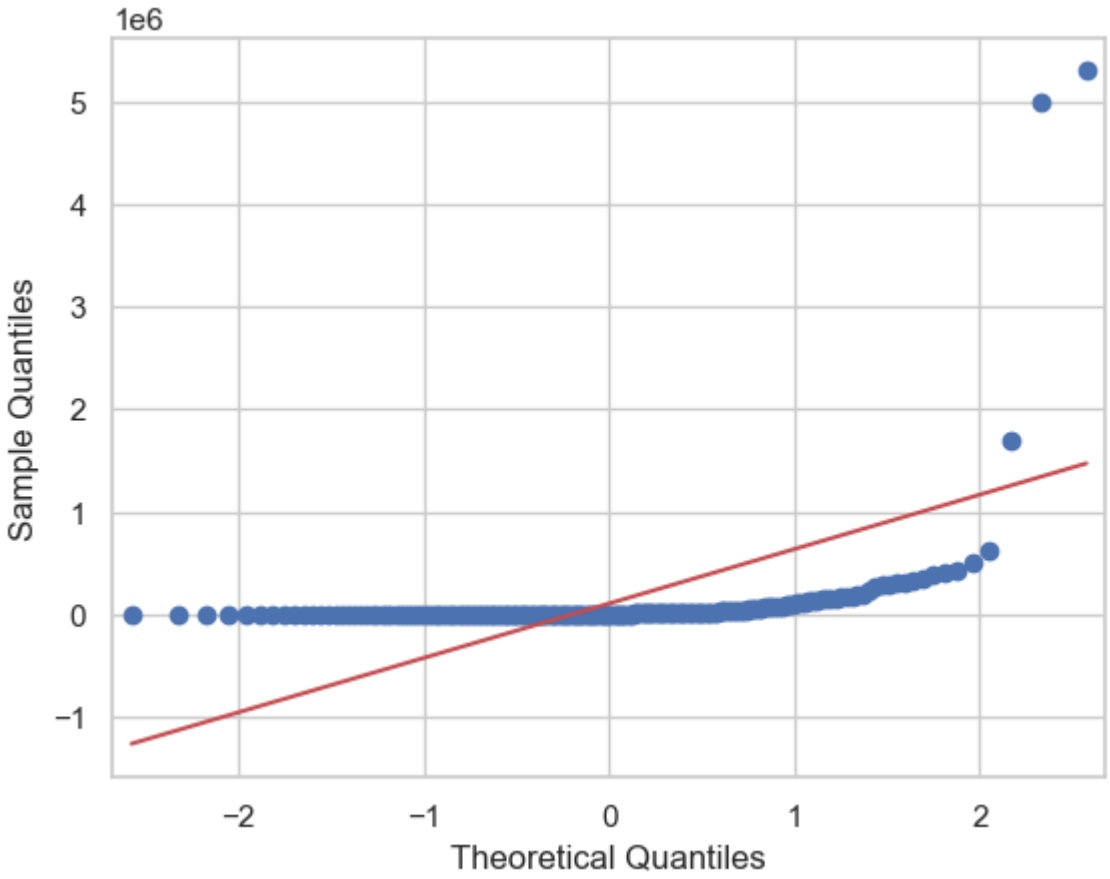
In [266]:
```
# Create scatter plots for pairs of numeric columns
numeric_columns = df.select_dtypes(include=['float64', 'int64'])
sns.pairplot(data=df, vars=numeric_columns.columns, diag_kind='kde')
plt.suptitle('Scatter Plots for Outlier Detection', y=1.02)
plt.show()
```



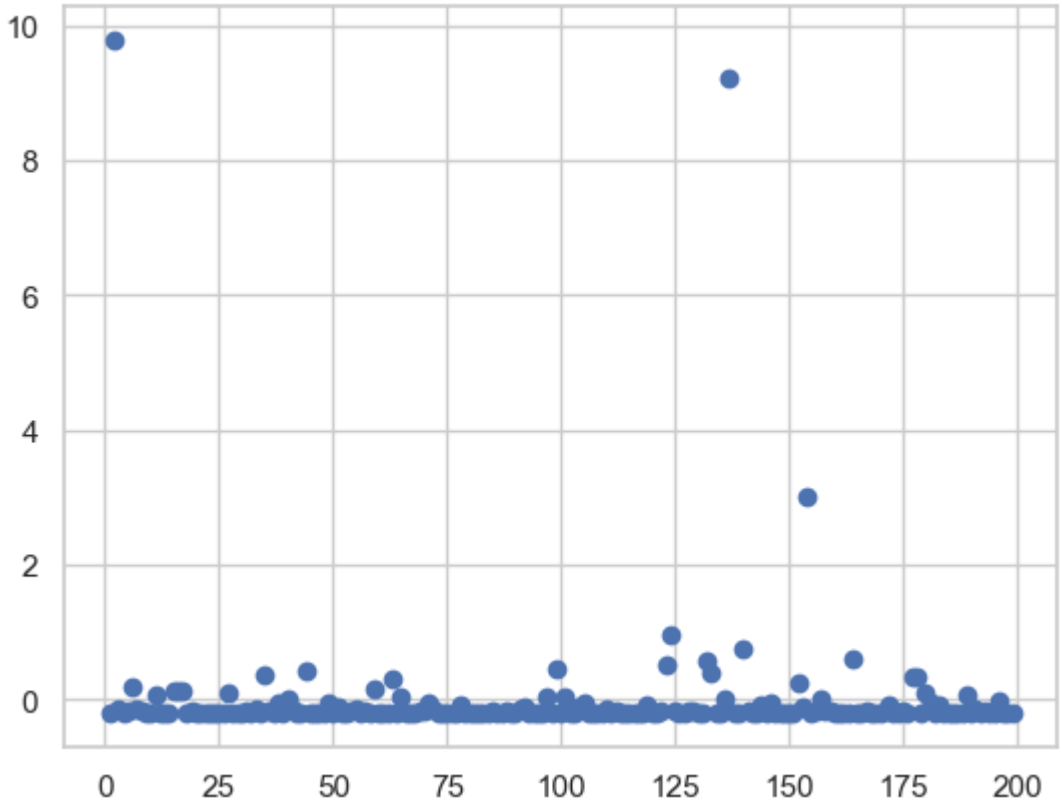
Outlier

In [267]:

```
from statsmodels.graphics.gofplots import qqplot

# Create a QQ plot
qqplot(df['Likes'], line='s')
plt.show()
```



In [270]:
```
from scipy import stats

z_scores = stats.zscore(df['Likes'])
plt.scatter(df['Rank'], z_scores)
plt.show()
```



Upper limit & Lower limit

In [279]:
```
upper_limit = df['Likes'].mean() + 3*df['Likes'].std()
lower_limit = df['Rank'].mean() + 3*df['Rank'].std()
print('upper limit:', upper_limit)
print('lower limit:', lower_limit)
```

```
upper limit: 1701275.3320238253
lower limit: 272.77152543170996
```

In [281]:
```
df.loc[(df['Likes'] < upper_limit) | (df['Likes'] > lower_limit)]
```

Out[281]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comı
0	1	tseries	4567	249500000	India	86200	2700	
1	2	MrBeast	4567	183500000	Estados Unidos	117400000	5300000	
2	3	CoComelon	4567	165500000	Unknown	7000000	24700	
3	4	SETIndia	4567	162600000	India	15600	166	
4	5	KidsDianaShow	4567	113500000	Unknown	3900000	12400	
...	...	...	...	...	...	...	...	
194	195	dianaandromahin8102	4567	26200000	Unknown	109000	322	
195	196	nickiminaj	4567	26100000	Estados Unidos	1600000	98300	
196	197	mariliamendoncareal	4567	26100000	Brasil	0	0	
197	198	TheLallantop	4567	26000000	India	30800	822	
198	199	AGT	4567	25900000	Estados Unidos	186800	3100	

199 rows × 9 columns

In [293]:
```
from pandas.api.types import CategoricalDtype
df['Categories'] = df['Categories'].astype( CategoricalDtype(categories
```

In [294]:
```
df['Categories'].unique()
```

Out[294]:
```
array([ True, False])
```

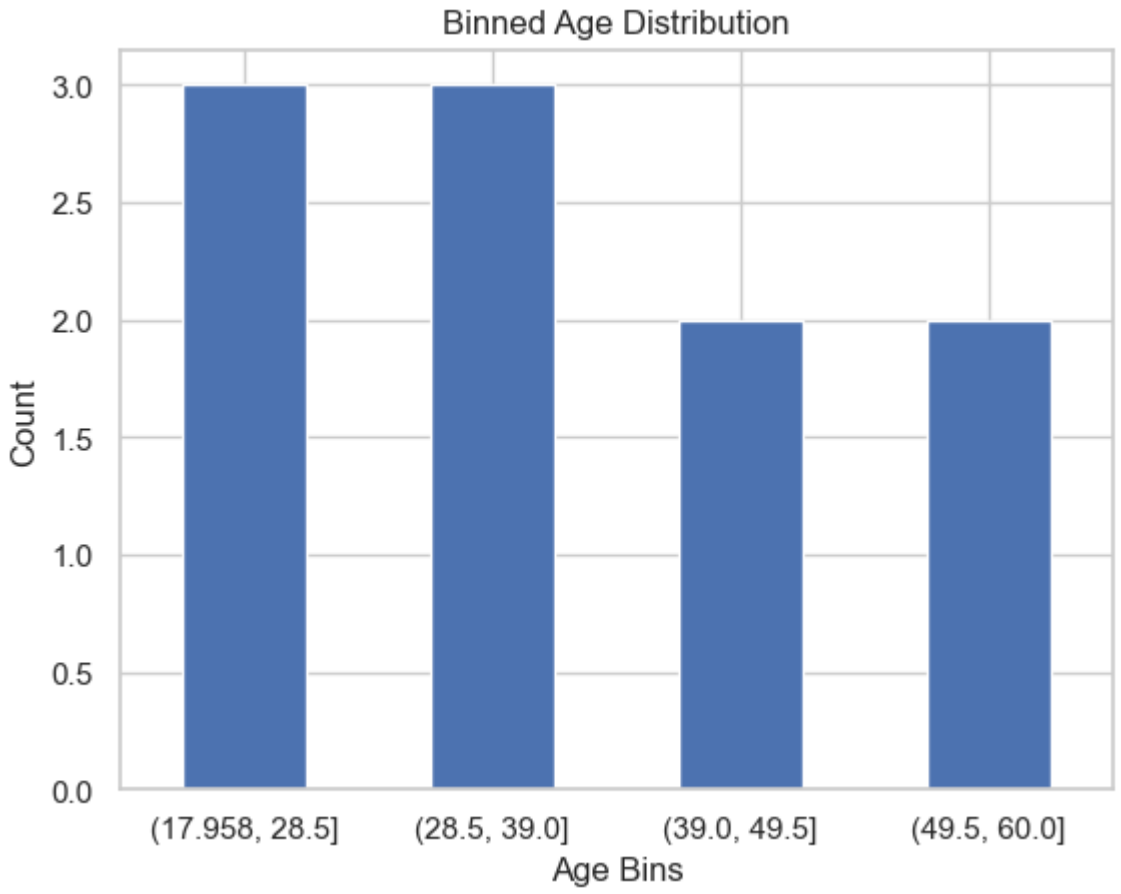
In [296]:
```
import pandas as pd

# Specify the number of bins or the bin edges
# For simplicity, let's assume you want to create three bins for a colu
num_bins = 3

# Perform data discretization
d_df=df['Likes'] = pd.cut(df['Comments'], bins=num_bins, labels=False)
d_df
```

Out[296]:
```
0      0
1      0
2      0
3      0
4      0
      ..
194    0
195    0
196    0
197    0
198    0
Name: Comments, Length: 199, dtype: int64
```

equal-width binning

In [297]:

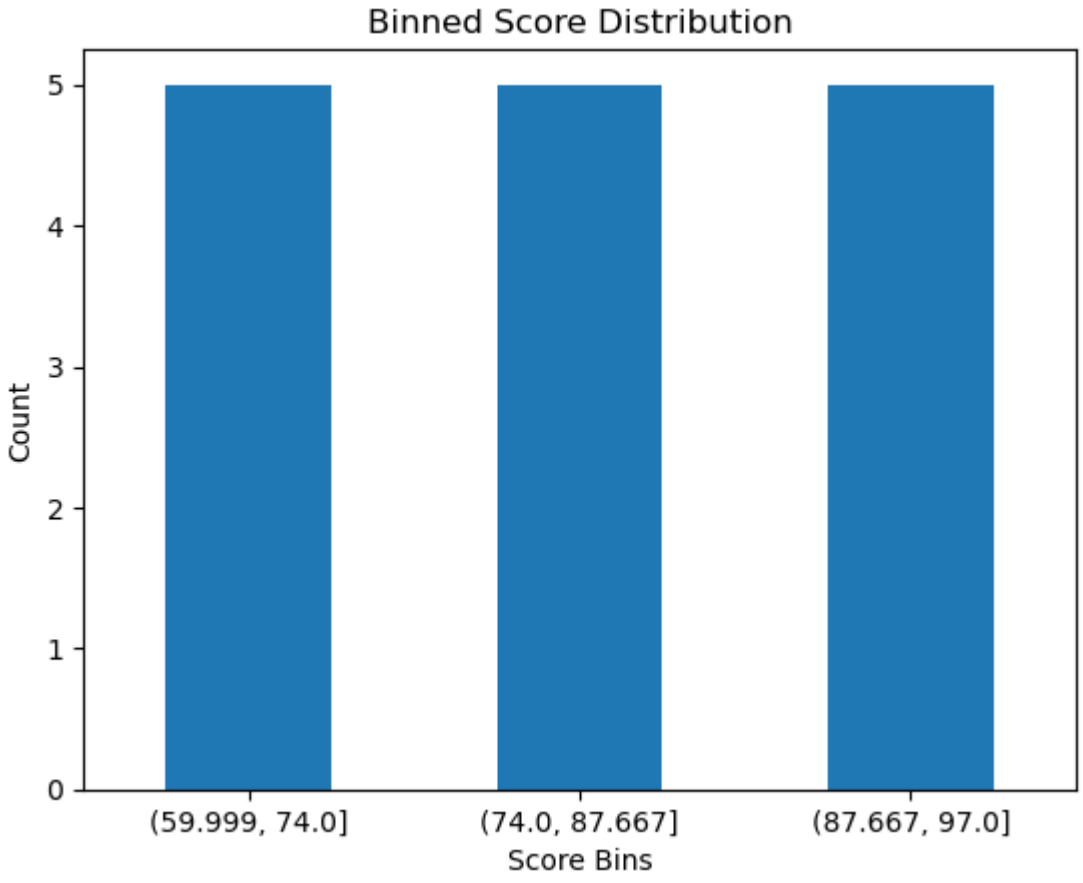
```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame with continuous data
data = {'Age': [25, 32, 45, 52, 60, 18, 35, 42, 28, 38]}
df = pd.DataFrame(data)

# Perform equal-width binning
num_bins = 4
df['Age_Binned'] = pd.cut(df['Age'], bins=num_bins)

# Count the number of data points in each bin
bin_counts = df['Age_Binned'].value_counts().sort_index()

# Create a bar graph
bin_counts.plot(kind='bar')
plt.xlabel('Age Bins')
plt.ylabel('Count')
plt.title('Binned Age Distribution')
plt.xticks(rotation=0)  # Ensure x-axis labels are not rotated
plt.show()
```



equal-frequency binning

In [4]:

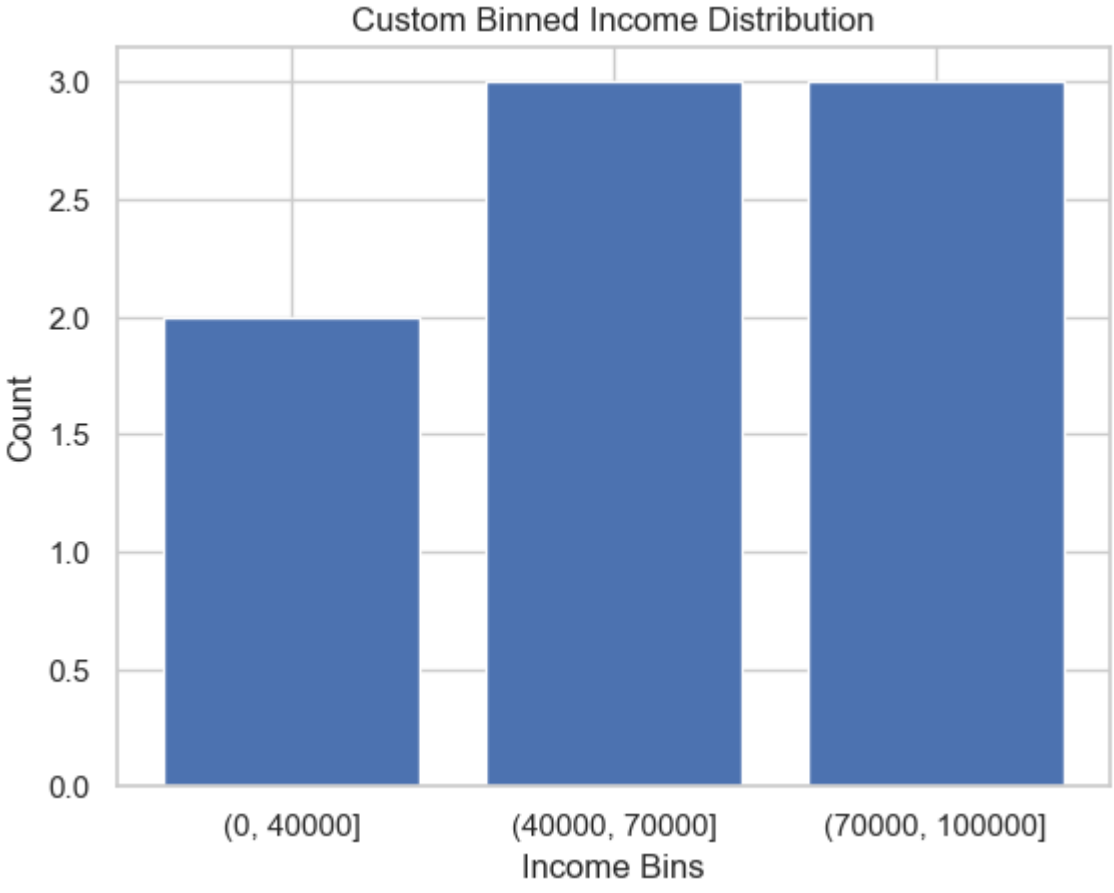
```
# import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame with continuous data
data = {'Score': [82, 91, 75, 68, 95, 87, 63, 78, 89, 72, 84, 97, 60, 9
df = pd.DataFrame(data)

# Define the number of bins
num_bins = 3

# Perform equal-frequency binning
df['Score_Binned'] = pd.qcut(df['Score'], q=num_bins)

# Count the number of data points in each bin
bin_counts = df['Score_Binned'].value_counts().sort_index()

# Create a bar graph
bin_counts.plot(kind='bar')
plt.xlabel('Score Bins')
plt.ylabel('Count')
plt.title('Binned Score Distribution')
plt.xticks(rotation=0)  # Ensure x-axis labels are not rotated
plt.show()
```



custom binning

In [299]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame with continuous data
data = {'Income': [25000, 35000, 45000, 55000, 65000, 75000, 85000, 950
df = pd.DataFrame(data)

# Define custom bin edges
custom_bins = [0, 40000, 70000, 100000]

# Perform custom binning
df['Income_Binned'] = pd.cut(df['Income'], bins=custom_bins)

# Count the number of data points in each bin
bin_counts = df['Income_Binned'].value_counts().sort_index()

# Create a histogram
plt.bar(bin_counts.index.astype(str), bin_counts.values)
plt.xlabel('Income Bins')
plt.ylabel('Count')
plt.title('Custom Binned Income Distribution')
plt.show()
```



cluster-based binning

In [353]:
```
1 pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\anusha v\anaconda3
\lib\site-packages (1.0.2)
Requirement already satisfied: joblib>=0.11 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-learn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\anusha v\a
naconda3\lib\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: numpy>=1.14.6 in c:\users\anusha v\anaconda
3\lib\site-packages (from scikit-learn) (1.21.5)
Requirement already satisfied: scipy>=1.1.0 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-learn) (1.9.1)
Note: you may need to restart the kernel to use updated packages.
```

In [354]:
```
1 pip install KMeans
```

```
Note: you may need to restart the kernel to use updated packages.

WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000019A50BAB310>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/kmeans/
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000019A50BAB640>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/kmeans/
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000019A50BAB940>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/kmeans/
WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000019A50BABAF0>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/kmeans/
WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'NewConnectionError('<pip._vendo
r.urllib3.connection.HTTPSConnection object at 0x0000019A50BABCA0>: Failed
to establish a new connection: [Errno 11001] getaddrinfo failed')': /simpl
e/kmeans/
ERROR: Could not find a version that satisfies the requirement KMeans (fro
m versions: none)
ERROR: No matching distribution found for KMeans
```

In [355]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a DataFrame with continuous data
data = {'Value': [12, 15, 18, 25, 35, 42, 55, 65]}
df = pd.DataFrame(data)

# Define the number of desired bins
num_bins = 3

# Create a K-means clustering model
kmeans = KMeans(n_clusters=num_bins, random_state=42)

# Fit the model to the data
df['Cluster'] = kmeans.fit_predict(df[['Value']])

# Calculate cluster centers
cluster_centers = kmeans.cluster_centers_

# Sort cluster centers to determine bin edges
cluster_centers.sort(axis=0)

# Create bins based on cluster centers
bin_edges = cluster_centers.flatten().tolist()
bin_edges.append(float('inf'))

# Perform cluster-based binning
df['Value_Binned'] = pd.cut(df['Value'], bins=bin_edges)

# Count the number of data points in each bin
bin_counts = df['Value_Binned'].value_counts().sort_index()

# Create a bar graph
bin_counts.plot(kind='bar')
plt.xlabel('Value Bins')
plt.ylabel('Count')
plt.title('Cluster-Based Binned Value Distribution')
plt.xticks(rotation=0)  # Ensure x-axis labels are not rotated
plt.show()
```

```
---------------------------------------------------------------------------
-
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_12064\3132253234.py in <module>
     10
     11 # Create a K-means clustering model
---> 12 kmeans = KMeans(n_clusters=num_bins, random_state=42)
     13
     14 # Fit the model to the data

NameError: name 'KMeans' is not defined
```

LabelEncoder

In [305]:
```
from sklearn.preprocessing import LabelEncoder

# Sample data
data = ['low', 'medium', 'high', 'low', 'high', 'medium']

# Create a label encoder
label_encoder = LabelEncoder()

# Fit the label encoder on the data and transform the data
encoded_data = label_encoder.fit_transform(data)

# The encoded data can now be used in your analysis or machine learning
print(encoded_data)

```

```
[1 2 0 1 0 2]
```

Handling outliers

In [364]:
```
import pandas as pd
from scipy import stats
column_name = 'Comments'
df['Z-score'] = stats.zscore(df["Comments"])
output_csv_file = 'z_scores_output.csv'
df.to_csv(output_csv_file, index=False)
print(f'Z-scores calculated and saved to {output_csv_file}')
```

```
Z-scores calculated and saved to z_scores_output.csv
```

In [363]:

```
import pandas as pd
Q1 = df['Likes'].quantile(0.2)
Q3 = df['Comments'].quantile(0.20)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Likes'] < lower_bound) | (df['Comments'] > upper_bou
outliers

```

Out[363]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comı
0	1	tseries	True	249500000	India	86200	2700	
1	2	MrBeast	False	183500000	Estados Unidos	117400000	5300000	
2	3	CoComelon	True	165500000	Unknown	7000000	24700	
3	4	SETIndia	False	162600000	India	15600	166	
4	5	KidsDianaShow	True	113500000	Unknown	3900000	12400	
...	...	...	...	...	...	...	...	
194	195	dianaandromahin8102	True	26200000	Unknown	109000	322	
195	196	nickiminaj	False	26100000	Estados Unidos	1600000	98300	
196	197	mariliamendoncareal	True	26100000	Brasil	0	0	
197	198	TheLallantop	False	26000000	India	30800	822	
198	199	AGT	True	25900000	Estados Unidos	186800	3100	

199 rows × 10 columns

In [321]:
```
import pandas as pd
df_no_duplicates = df.drop_duplicates()
df_no_duplicates.to_csv('cleaned_data.csv', index=False)
df_no_duplicates
```

Out[321]:

	Rank	Username	Categories	Suscribers	Country	Visits	Likes	Comı
0	1	tseries	True	249500000	India	86200	2700	
1	2	MrBeast	False	183500000	Estados Unidos	117400000	5300000	
2	3	CoComelon	True	165500000	Unknown	7000000	24700	
3	4	SETIndia	False	162600000	India	15600	166	
4	5	KidsDianaShow	True	113500000	Unknown	3900000	12400	
...	...	...	...	...	...	...	...	
194	195	dianaandromahin8102	True	26200000	Unknown	109000	322	
195	196	nickiminaj	False	26100000	Estados Unidos	1600000	98300	
196	197	mariliamendoncareal	True	26100000	Brasil	0	0	
197	198	TheLallantop	False	26000000	India	30800	822	
198	199	AGT	True	25900000	Estados Unidos	186800	3100	

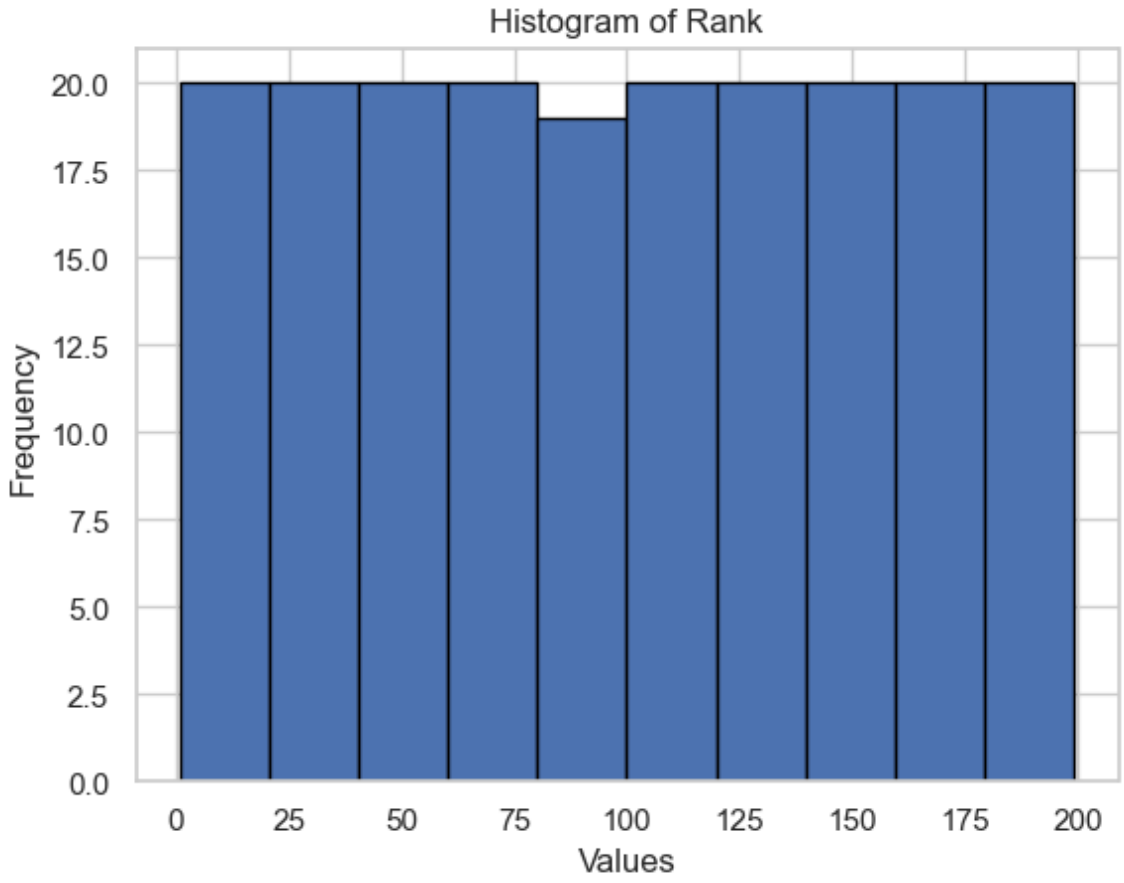
199 rows × 10 columns

Univariate analysis

In [326]:

```
import pandas as pd
import matplotlib.pyplot as plt

selected_column = 'Rank' # Replace 'column_name' with the name of the c
mean = df["Likes"].mean()
median = df["Comments"].median()
mode = df["Visits"].mode().iloc[0]
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode}")
plt.hist(df["Rank"], bins=10, edgecolor='black')
plt.xlabel("Values")
plt.ylabel('Frequency')
plt.title(f'Histogram of {selected_column}')
plt.show()

```

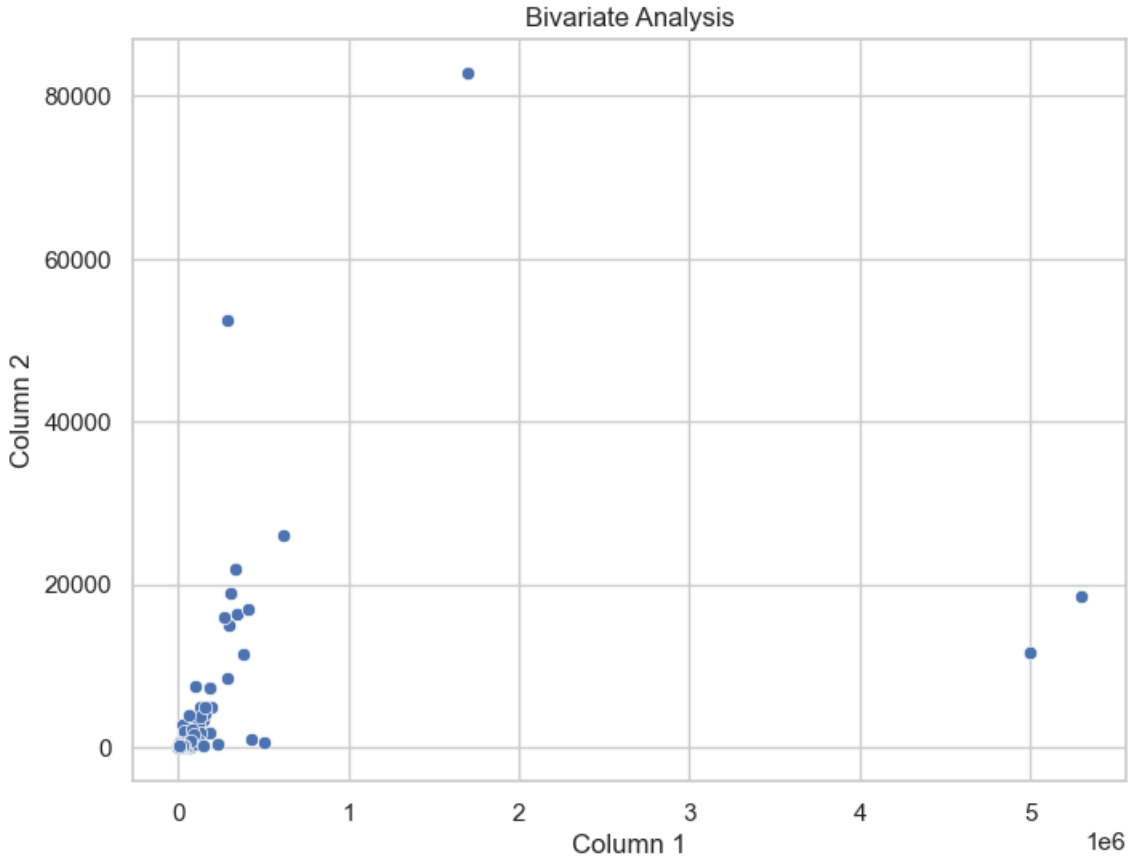
```
Mean: 104774.6783919598
Median: 28.0
Mode: 0
```



Bivariate Analysis

In [327]:

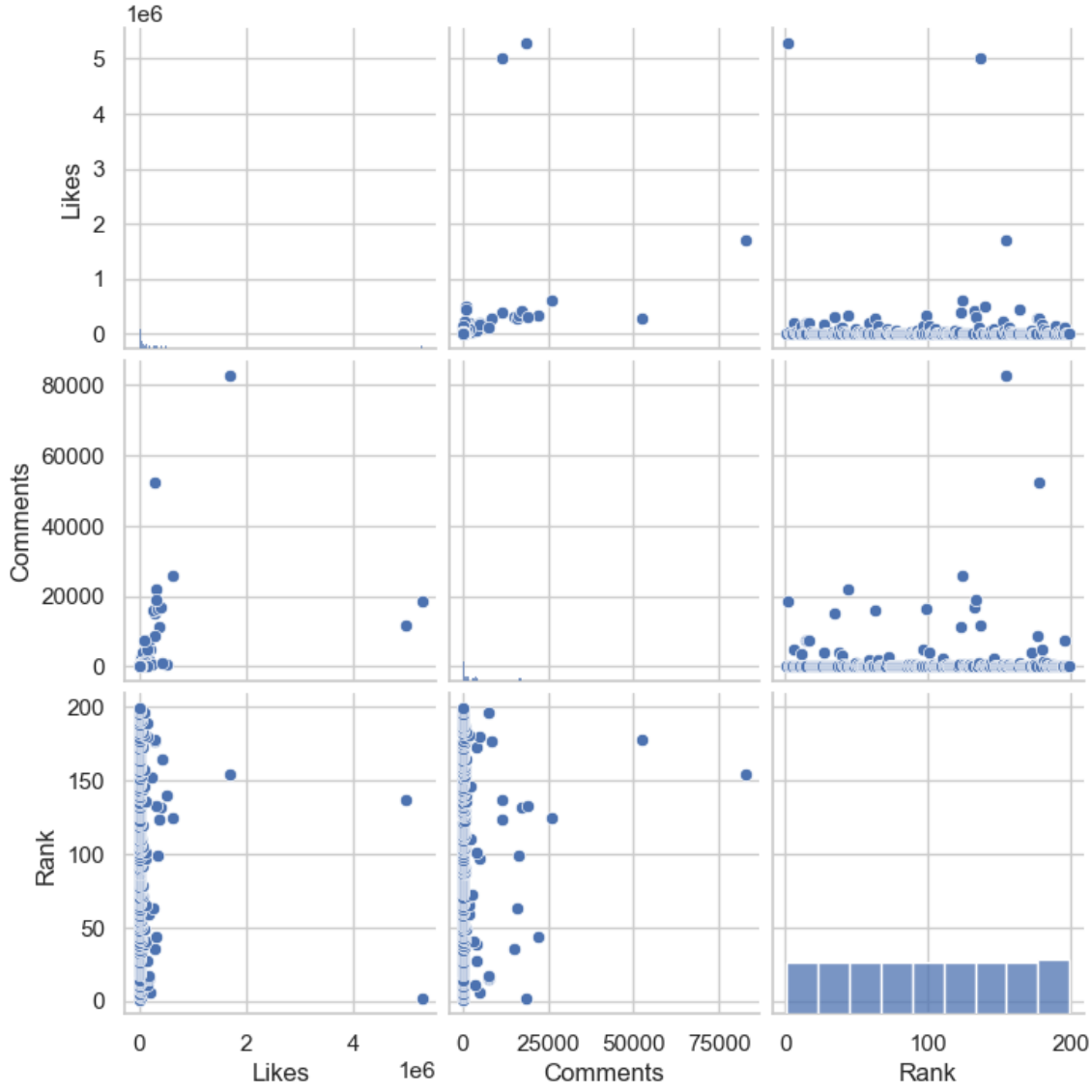
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Read the CSV file into a DataFrame

# Select two columns for analysis (replace 'column1' and 'column2' with
column1 = df['Likes']
column2 = df['Comments']
# Create scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=column1, y=column2)
plt.title('Bivariate Analysis')
plt.xlabel('Column 1')
plt.ylabel('Column 2')
plt.show()
```



Multivariate analysis

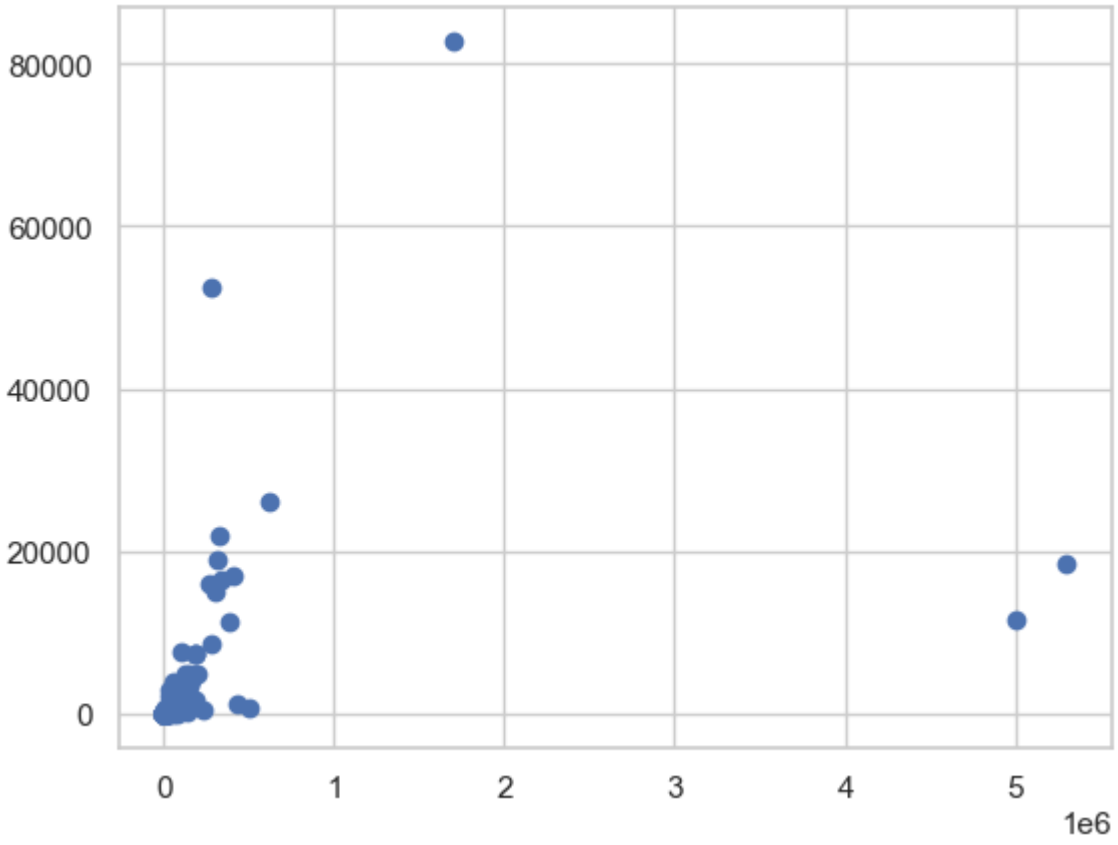
In [328]:

```
import pandas as pd
import seaborn as sns
numeric_data = df[['Likes', 'Comments', 'Rank']]
sns.pairplot(numeric_data)
plt.show()
```



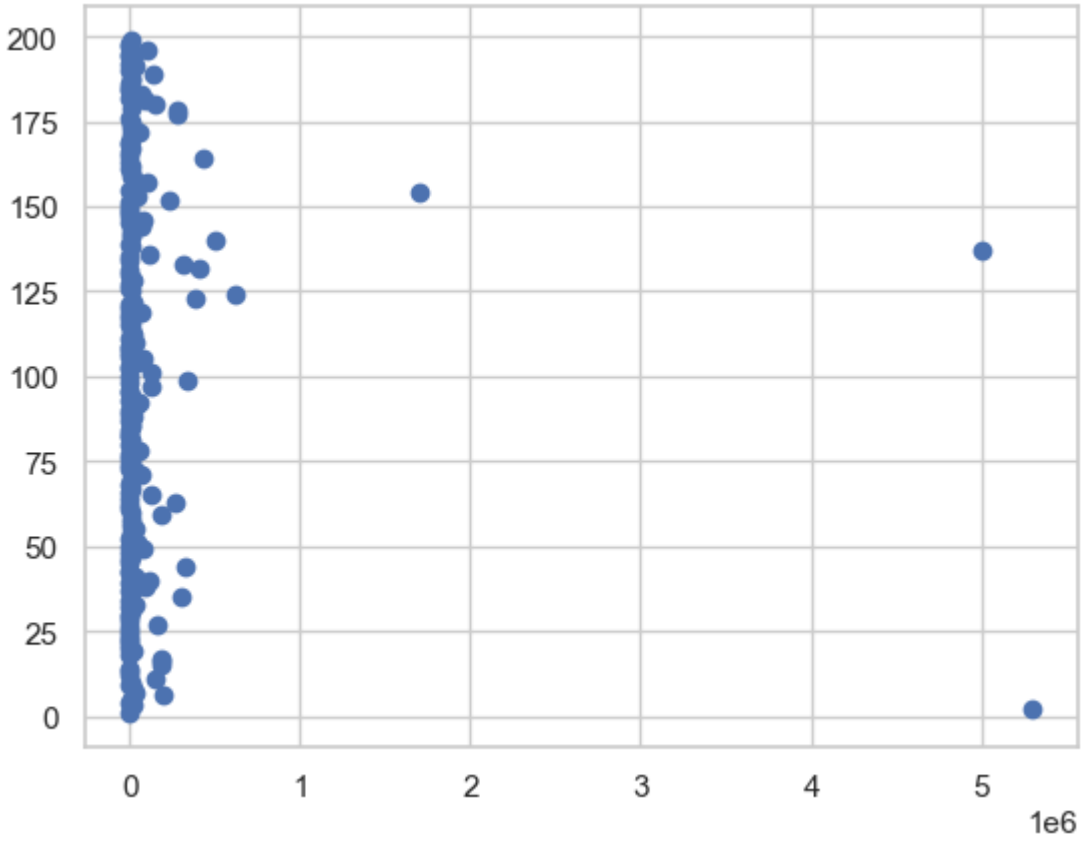
In [332]:
```
1 import matplotlib.pyplot as plt
2 %matplotlib inline
3 plt.scatter(df['Likes'],df['Comments'])
```

Out[332]: <matplotlib.collections.PathCollection at 0x242f0c7cc10>



In [335]:
```
1 plt.scatter(df["Likes"],df["Rank"])
```

Out[335]:
```
<matplotlib.collections.PathCollection at 0x242ee336bb0>
```



X-Test Y-Test

In [336]:
```
1 x = df[['Likes','Comments']]
2 y = df[['Rank']]
3 x
```

Out[336]:

	Likes	Comments
0	2700	78
1	5300000	18500
2	24700	0
3	166	9
4	12400	0
...	...	...
194	322	0
195	98300	7600
196	0	0
197	822	52
198	3100	197

199 rows × 2 columns

In [337]:
```
1 x
```

Out[337]:

	Likes	Comments
0	2700	78
1	5300000	18500
2	24700	0
3	166	9
4	12400	0
...	...	...
194	322	0
195	98300	7600
196	0	0
197	822	52
198	3100	197

199 rows × 2 columns

In [338]:
```
1 y
```

Out[338]:

	Rank
0	1
1	2
2	3
3	4
4	5
...	...
194	195
195	196
196	197
197	198
198	199

199 rows × 1 columns

In [339]:
```
1 from sklearn.model_selection import train_test_split
2 x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
```

In [340]:
```
1 len(x_train)
```

Out[340]: 159

In [341]:
```
1 len(x_test)
```

Out[341]: 40

In [342]:
```
1 x_train
```

Out[342]:

	Likes	Comments
57	7500	0
193	1800	20
51	430	21
25	2600	0
187	4300	17
...	...	...
190	37700	192
136	5000000	11600
163	431100	1100
2	24700	0
125	1800	73

159 rows × 2 columns

In [343]:
```
1 y_train
```

Out[343]:

	Rank
57	58
193	194
51	52
25	26
187	188
...	...
190	191
136	137
163	164
2	3
125	126

159 rows × 1 columns

In [344]:
```
1 x_test
```

Out[344]:

	Likes	Comments
79	433	22
92	255	3
181	62	2
173	6600	0
31	2100	33
3	166	9
63	994	9
39	117100	3000
49	180	2
83	59	0
102	641	0
50	39800	593
56	8200	205
101	601	27
81	1800	0
189	471	24
158	8500	275
122	382000	11400
192	21300	214
14	180300	7400
84	10800	0
120	83	1
26	156500	4200
68	13100	0
124	13200	386
38	163	7
121	23300	504
156	106500	428
180	85800	1600
64	128900	1800
46	3400	0
12	615	21
76	211	9
172	6300	200
5	197300	4900
107	267	6
191	993	127
65	184	0
7	22100	0

	Likes	Comments
117	88	0

In [345]:
```
1 y_test
```

Out[345]:

	Rank
79	80
92	93
181	182
173	174
31	32
3	4
63	64
39	40
49	50
83	84
102	103
50	51
56	57
101	102
81	82
189	190
158	159
122	123
192	193
14	15
84	85
120	121
26	27
68	69
124	125
38	39
121	122
156	157
180	181
64	65
46	47
12	13
76	77
172	173
5	6
107	108
191	192
65	66
7	8

	Rank
117	118

In [346]:
```
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
clf.fit(x_train,y_train)
```

Out[346]:
```
LinearRegression()
```

In [347]:
```
clf.predict(x_test)
```

Out[347]:
```
array([[101.30787323],
       [101.29595555],
       [101.29674181],
       [101.24501668],
       [101.30273441],
       [101.30083679],
       [101.29446324],
       [102.49251971],
       [101.29583351],
       [101.29536619],
       [101.29088623],
       [101.40417926],
       [101.3760694 ],
       [101.31007684],
       [101.2819648 ],
       [101.30897944],
       [101.42271533],
       [106.32806798],
       [101.28152608],
       [105.0832193 ],
       [101.21268709],
       [101.29588081],
       [103.02846903],
       [101.19498278],
       [101.46416591],
       [101.29946116],
       [101.46894538],
       [100.77536008],
       [101.75434849],
       [101.56245719],
       [101.26964876],
       [101.30577292],
       [101.3004904 ],
       [101.3871979 ],
       [103.20396193],
       [101.29796126],
       [101.37699539],
       [101.294404  ],
       [101.12570507],
       [101.29514296]])
```

In [348]:
```
clf.score(x_test, y_test)
```

Out[348]:
```
-0.03599704198236631
```

In [349]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199 entries, 0 to 198
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Rank        199 non-null    int64  
 1   Username    199 non-null    object 
 2   Categories  199 non-null    bool   
 3   Suscribers  199 non-null    int64  
 4   Country     199 non-null    object 
 5   Visits      199 non-null    int64  
 6   Likes       199 non-null    int64  
 7   Comments    199 non-null    int64  
 8   Links       199 non-null    object 
 9   Z-score     199 non-null    float64
dtypes: bool(1), float64(1), int64(5), object(3)
memory usage: 14.3+ KB
```

In [350]:
```
1 x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,r
2 x_test
```

Out[350]:

	Likes	Comments
59	4600	0
5	197300	4900
20	413	3
124	13200	386
52	7500	177
19	1100	4
161	5700	0
55	5700	0
69	10100	0
2	24700	0
98	341800	16500
10	146900	3400
75	3	0
134	1400	67
193	1800	20
63	994	9
110	442	8
78	3400	0
178	4100	305
114	623	0
149	1200	33
129	508	0
61	2400	0
87	20100	363
102	641	0
120	83	1
168	0	0
1	5300000	18500
47	242	13
171	58500	4000
185	2200	176
39	117100	3000
76	211	9
91	54100	214
35	10200	345
121	23300	504
169	3100	0
162	242	9
46	3400	0

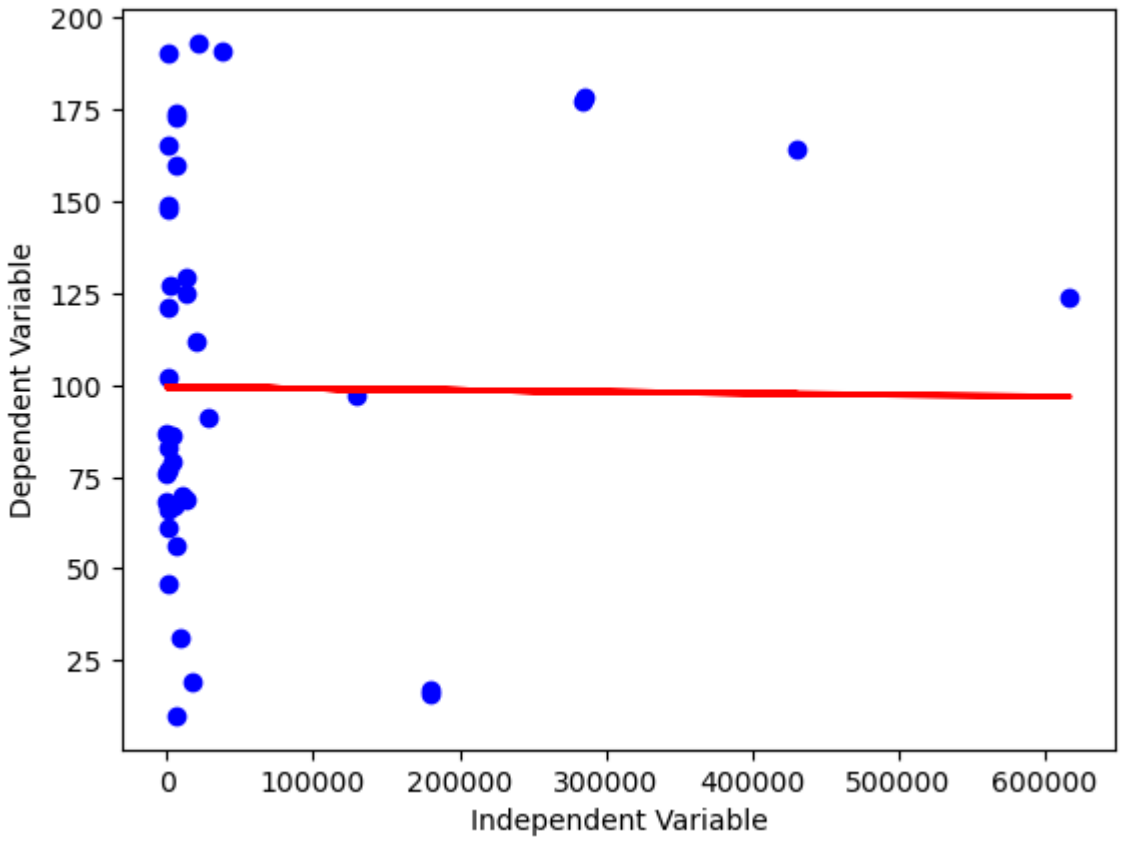
	Likes	Comments
173	6600	0
189	471	24
7	22100	0
26	156500	4200
148	798	0
58	183400	1800
72	255	9
103	42400	541
198	3100	197
56	8200	205
184	651	0
24	231	9
43	330400	22000
101	601	27
142	7600	108
21	392	18
60	1100	30
174	8700	668
70	73500	1600
90	27500	357
49	180	2

Simple Linear Regression

In [3]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

X = df[['Likes']]  # Independent variable
y = df['Rank']  # Dependent variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.xlabel('Independent Variable')
plt.ylabel('Dependent Variable')
plt.show()


```

```
Mean Squared Error: 2893.7832522878493
```



Multiple Linear Regression

In [36]:

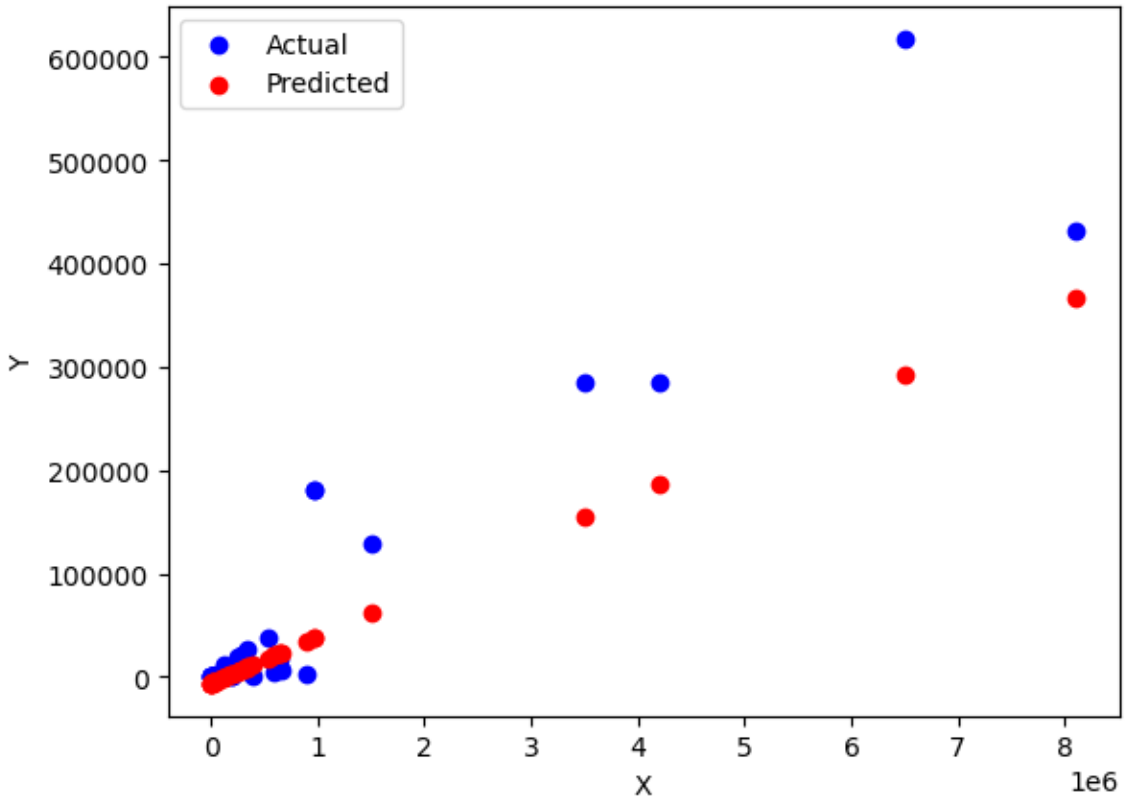
```python
# Import the necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Create or load a dataset (example data)
data = {
    'Feature1': [1.2, 2.2, 3.1, 4.5, 5.0],
    'Feature2': [2.4, 3.9, 4.8, 5.9, 7.0],
    'Feature3': [1.0, 2.0, 3.0, 4.0, 5.0],
    'Target': [2.3, 3.6, 5.0, 5.5, 7.8]
}

df = pd.DataFrame(data)

# Split the data into training and testing sets
X = df[['Feature1', 'Feature2', 'Feature3']]
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

print("Coefficients:", coefficients)
print("Intercept:", intercept)

```

```
Mean Squared Error: 0.04
R-squared: nan
Coefficients: [-2.   -0.5   3.85]
Intercept: 2.049999999999978

C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\metrics\_regression.
py:796: UndefinedMetricWarning: R^2 score is not well-defined with less th
an two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```

polynomial Linear Regression

In [45]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt


X = df[['Visits']]  # Features (input variable)
y = df['Likes']  # Target variable (output variable)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

degree = 2  # Set the degree of the polynomial
poly = PolynomialFeatures(degree=degree)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

model = LinearRegression()
model.fit(X_train_poly, y_train)

y_pred = model.predict(X_test_poly)

# Plot the actual vs. predicted values
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, y_pred, color='red', label='Predicted')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()

```



Loading [MathJax]/extensions/MathML/physics.js

Cross Validation

In [49]:

```python
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
from sklearn.svm import SVC

# Load your dataset (replace this with your data)
data = load_iris()
X = data.data
y = data.target

# Set the number of folds (k)
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Initialize a list to store the accuracy scores for each fold
accuracy_scores = []

# Iterate over each fold
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Create and train your machine learning model (replace with your m
    model = SVC()
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Calculate the accuracy for this fold
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)

# Calculate the mean and standard deviation of the accuracy scores
mean_accuracy = np.mean(accuracy_scores)
std_accuracy = np.std(accuracy_scores)

# Print the results
print(f"Accuracy scores for {k}-fold cross-validation: {accuracy_scores
print(f"Mean Accuracy: {mean_accuracy:.2f}")
print(f"Standard Deviation: {std_accuracy:.2f}")

```

```
Accuracy scores for 5-fold cross-validation: [1.0, 1.0, 0.933333333333333
3, 0.9333333333333333, 0.9666666666666667]
Mean Accuracy: 0.97
Standard Deviation: 0.03
```

Evaluation regression model

In [37]:

```
# Import the necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, explained_var

# Create or load a dataset (example data)
data = {
    'Feature1': [1.2, 2.2, 3.1, 4.5, 5.0],
    'Feature2': [2.4, 3.9, 4.8, 5.9, 7.0],
    'Feature3': [1.0, 2.0, 3.0, 4.0, 5.0],
    'Target': [2.3, 3.6, 5.0, 5.5, 7.8]
}

df = pd.DataFrame(data)

# Split the data into training and testing sets
X = df[['Feature1', 'Feature2', 'Feature3']]
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)  # Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared (coefficient of determinatio
explained_variance = explained_variance_score(y_test, y_pred)  # Explai
mae = mean_absolute_error(y_test, y_pred)  # Mean Absolute Error

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
print(f"Explained Variance Score: {explained_variance:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")

```

```
Mean Squared Error: 0.04
R-squared: nan
Explained Variance Score: 1.00
Mean Absolute Error: 0.20

C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\metrics\_regression.
py:796: UndefinedMetricWarning: R^2 score is not well-defined with less th
an two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```

Random Forest Regressor

In [4]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
X, y = make_regression(n_features=4, n_informative=2, random_state=0, s
rfr = RandomForestRegressor(max_depth=3)
rfr.fit(X, y)
print(rfr.predict([[0, 1, 0, 1]]))
```

```
[32.85592054]
```

Lasso Regression

In [15]:
```
from sklearn import linear_model
import numpy as np
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16))
# Fit regression model
lassoReg = linear_model.Lasso(alpha=0.1)
lassoReg.fit(X,y)
# Predict
X_test = np.arange(0.0, 5.0, 1)[:, np.newaxis]
lassoReg.predict(X_test)

```

Out[15]:
```
array([ 0.78305084,  0.49957596,  0.21610108, -0.0673738 , -0.35084868])
```

In [9]:
```
X
```

Out[9]:
```
array([[5.71874087e-04],
       [9.14413867e-02],
       [9.68347894e-02],
       [1.36937966e-01],
       [1.95273916e-01],
       [2.49767295e-01],
       [2.66812726e-01],
       [4.25221057e-01],
       [4.61692974e-01],
       [4.91734169e-01],
       [5.11672144e-01],
       [5.16130033e-01],
       [6.50142861e-01],
       [6.87373521e-01],
       [6.96381736e-01],
       [7.01934693e-01],
       [7.33642875e-01],
       [7.33779454e-01],
       [8.26770986e-01],
```

Loading [MathJax]/extensions/MathML/content-mathml.js

In [11]:
```
1 y
```

Out[11]:
```
array([ 5.71874056e-04,  9.13140084e-02,  9.66835240e-02,  1.36510390e-01,
        1.94035253e-01,  2.47178482e-01,  2.63658285e-01,  4.12522163e-01,
        4.45464463e-01,  4.72155294e-01,  4.89635918e-01,  4.93517994e-01,
        6.05300129e-01,  6.34509327e-01,  6.41446076e-01,  6.45696215e-01,
        6.69579781e-01,  6.69681218e-01,  7.35748344e-01,  7.50720534e-01,
        8.02397078e-01,  8.36304301e-01,  8.53289303e-01,  8.71444956e-01,
        9.70605329e-01,  9.85824622e-01,  9.91311217e-01,  9.94728365e-01,
        9.98252126e-01,  9.99993246e-01,  9.99977003e-01,  9.87699652e-01,
        9.85915646e-01,  9.15904395e-01,  9.14069859e-01,  8.77830014e-01,
        8.77534605e-01,  8.70630546e-01,  8.69934082e-01,  8.65235592e-01,
        8.60400242e-01,  7.84646075e-01,  6.31686553e-01,  5.37227991e-01,
        4.58019714e-01,  4.45838347e-01,  4.32721158e-01,  3.41153283e-01,
        2.67699531e-01,  2.07303185e-01,  1.93830272e-01, -1.76451855e-01,
       -2.09188375e-01, -2.49907779e-01, -2.80682239e-01, -2.86826010e-01,
       -3.12470754e-01, -3.14585948e-01, -3.24429114e-01, -3.49651545e-01,
       -4.43974821e-01, -4.69211146e-01, -5.64011437e-01, -5.72153265e-01,
       -5.74888492e-01, -7.20694771e-01, -7.59230658e-01, -7.80436587e-01,
       -8.58088588e-01, -9.45898629e-01, -9.48667092e-01, -9.48706725e-01,
       -9.71490934e-01, -9.80974083e-01, -9.85684166e-01, -9.97200340e-01,
       -9.99943982e-01, -9.97032450e-01, -9.91701462e-01, -9.73227696e-0
1])
```

Support Vector Regression

In [2]:
```
1  from sklearn.svm import SVR
2  import numpy as np
3  rng = np.random.RandomState(1)
4  X = np.sort(5 * rng.rand(80, 1), axis=0)
5  y = np.sin(X).ravel()
6  y[::5] += 3 * (0.5 - rng.rand(16))
7  # Fit regression model
8  svr = SVR().fit(X, y)
9  # Predict
10 X_test = np.arange(0.0, 5.0, 1)[:, np.newaxis]
11 svr.predict(X_test)
12
```

Out[2]:
```
array([-0.07840308,  0.78077042,  0.81326895,  0.08638149, -0.6928019 ])
```

In [44]:

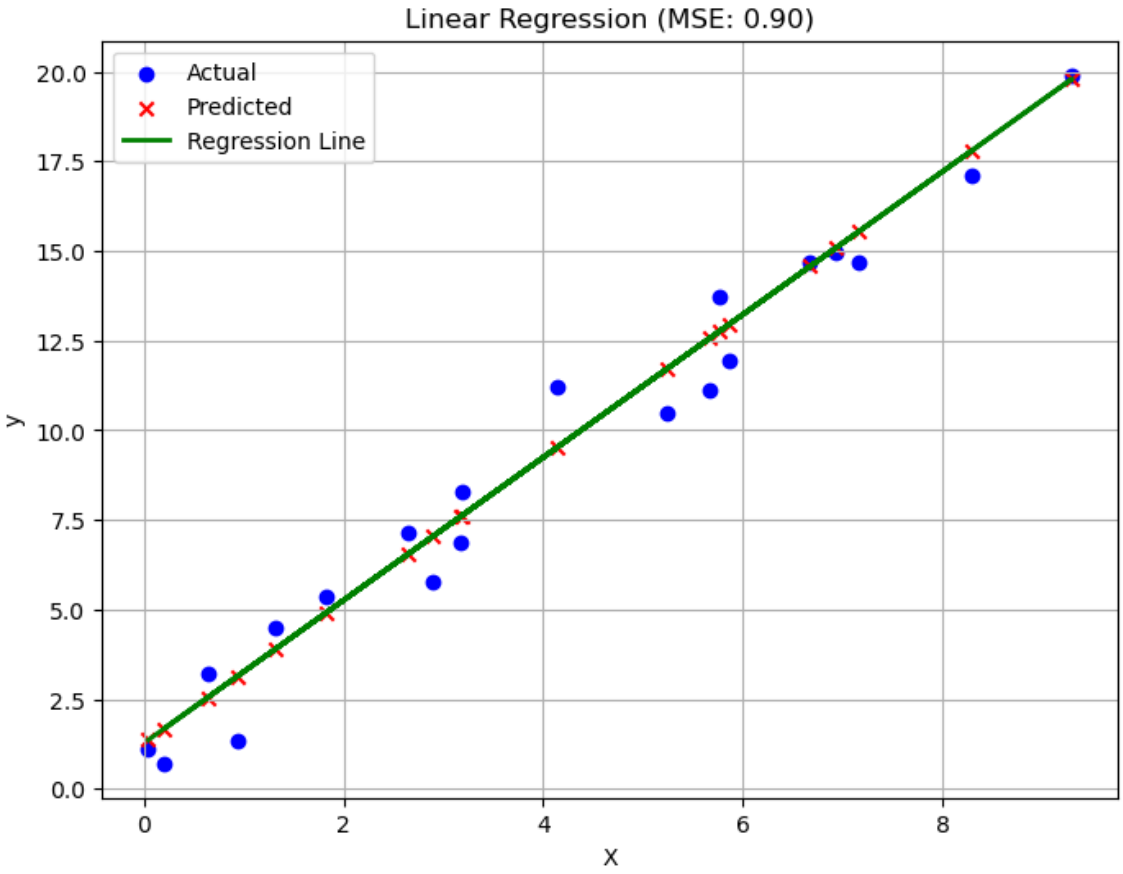
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
# Load your dataset from a CSV file
df = pd.read_csv (r"C:\Users\Anusha V\Desktop\YouTube.csv")
# Split the data into features (X) and the target variable (y)
X = data[['Likes']] # Replace 'feature_column' with your feature column
y = data['Rank'] # Adjust 'target_column' to your target variable
# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
# Create and train the Support Vector Regression model
model = SVR(kernel='linear') # You can choose the kernel type (e.g., 'l
model.fit(X_train, y_train)
# Make predictions on the testing set
y_pred = model.predict(X_test)
# Evaluate the model's performance using regression metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
# Plot the true values and predicted values
plt.scatter(X_test, y_test, color='b', label='True')
plt.scatter(X_test, y_pred, color='r', label='Predicted')
plt.title('SVR - True vs. Predicted')
plt.xlabel('Feature')
plt.ylabel('Target')
plt.legend(loc='upper left')
plt.show()
print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'R-squared (R²): {r2:.4f}')

```

```
  File "C:\Users\Anusha V\AppData\Local\Temp\ipykernel_8816\2609282417.p
y", line 15
    model = SVR(kernel='linear') # You can choose the kernel type (e.g.,
'line
    ^
SyntaxError: positional argument follows keyword argument
```

linear regression model

In [28]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = X[:80], X[80:], y[:80], y[80:]

# Fit a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Calculate the overall MSE
mse = mean_squared_error(y_test, predictions)

# Create a scatter plot for the linear regression
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, predictions, color='red', marker='x', label='Predic
plt.plot(X_test, predictions, color='green', linewidth=2, label='Regres
plt.xlabel('X')
plt.ylabel('y')
plt.title(f'Linear Regression (MSE: {mse:.2f})')
plt.legend()
plt.grid()
plt.show()

```



Mean Absolute Error(MAE)

In [13]:
```
# from sklearn.metrics import mean_absolute_error

# Actual values (ground truth)
actual = [10, 20, 30, 40, 50]

# Predicted values from your model
predicted = [12, 18, 28, 37, 45]

# Calculate the Mean Absolute Error
mae = mean_absolute_error(actual, predicted)

print(f"Mean Absolute Error: {mae}")
```

```
Mean Absolute Error: 2.8
```

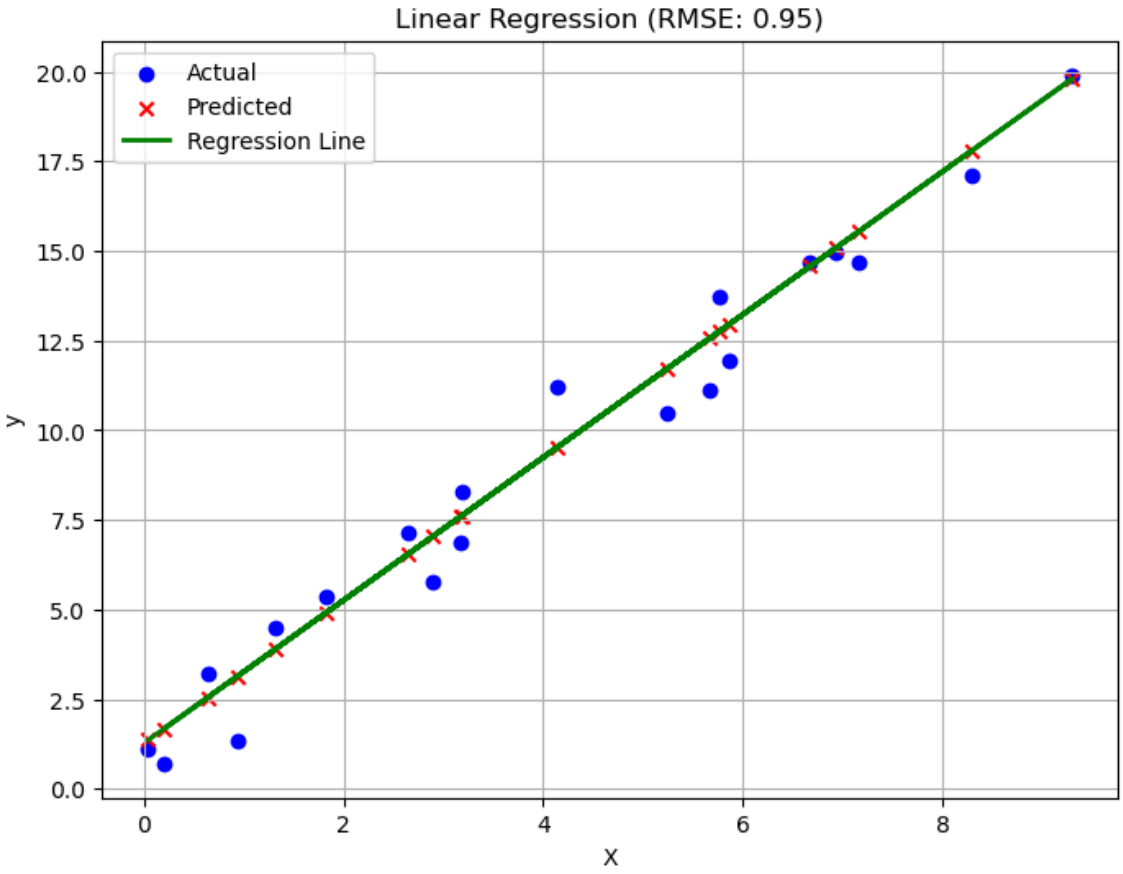
In [12]:
```
import pandas as pd
from sklearn.metrics import mean_absolute_error
df = pd.read_csv (r"C:\Users\Anusha V\Desktop\YouTube.csv")
actual = df['Likes']
predicted = df['Rank']
mae = mean_absolute_error(actual, predicted)
print(f"Mean Absolute Error (MAE): {mae}")
```

```
Mean Absolute Error (MAE): 104689.26130653266
```

root mean squared error

In [33]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = X[:80], X[80:], y[:80], y[80:]

# Fit a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Calculate squared errors for each prediction
squared_errors = (predictions - y_test) ** 2

# Calculate the overall RMSE
rmse = np.sqrt(mean_squared_error(y_test, predictions))

# Create a scatter plot for the linear regression with RMSE
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, predictions, color='red', marker='x', label='Predic
plt.plot(X_test, predictions, color='green', linewidth=2, label='Regres
plt.xlabel('X')
plt.ylabel('y')
plt.title(f'Linear Regression (RMSE: {rmse:.2f})')
plt.legend()
plt.grid()
plt.show()
```



In [9]:

```python
import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt

# Actual values
actual = [10, 12, 15, 18, 20]

# Predicted values
predicted = [9, 11, 14, 17, 19]

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(actual, predicted)

# Calculate RMSE
rmse = sqrt(mse)

print("RMSvE:", rmse)
```

```
RMSE: 1.0
```

R-squared (R2) score

In [49]:

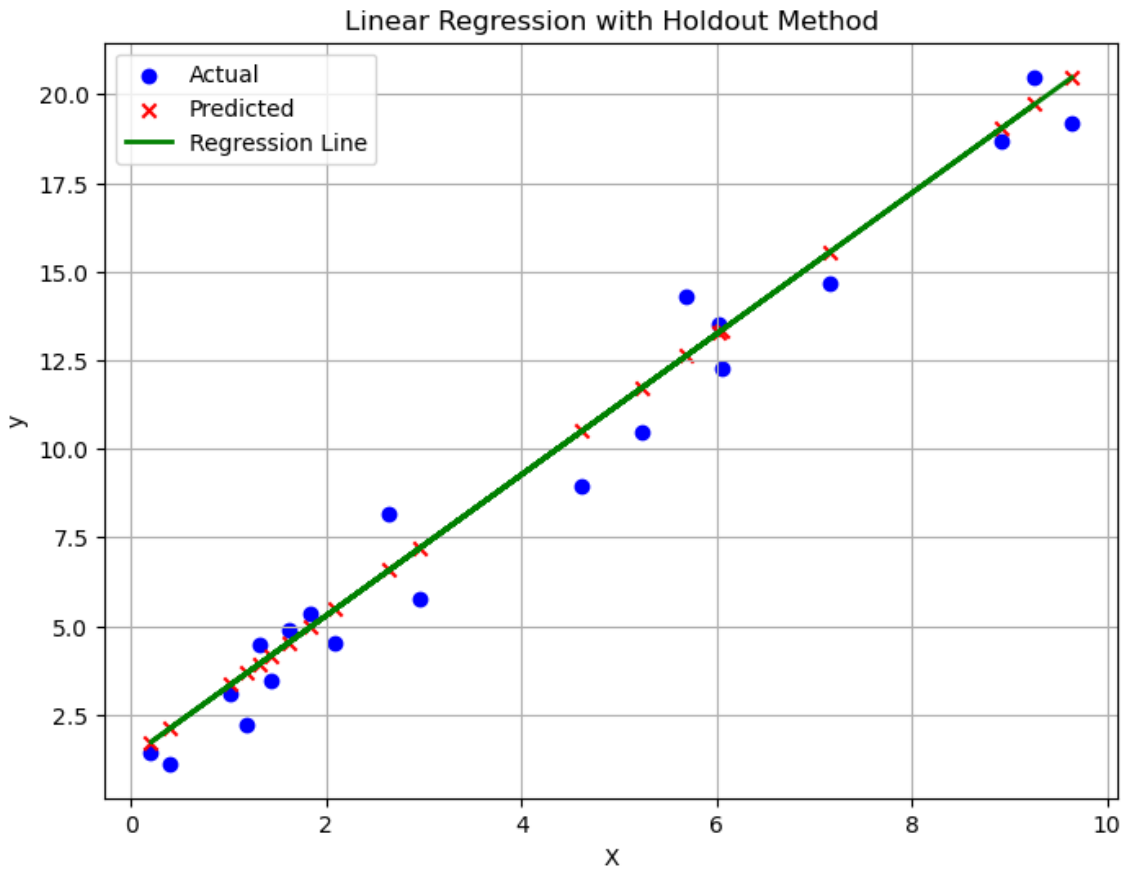
```
from sklearn.metrics import r2_score

# Assuming you have observed actual and predicted values
actual_values = [10, 20, 30, 40, 50]
predicted_values = [12, 18, 28, 38, 52]

# Calculate the R-squared (R2) score
r2 = r2_score(actual_values, predicted_values)

print(f"R-squared (R2) Score: {r2}")
```

```
R-squared (R2) Score: 0.98
```

holdout method

In [35]:

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1)

# Split the data into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Fit a linear regression model on the training data
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Create a scatter plot for the linear regression
plt.figure(figsize=(8, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.scatter(X_test, predictions, color='red', marker='x', label='Predic
plt.plot(X_test, predictions, color='green', linewidth=2, label='Regres
plt.xlabel('X')
plt.ylabel('y')
plt.title('Linear Regression with Holdout Method')
plt.legend()
plt.grid()
plt.show()
```



Loading [MathJax]/extensions/MathML/content-mathml.js

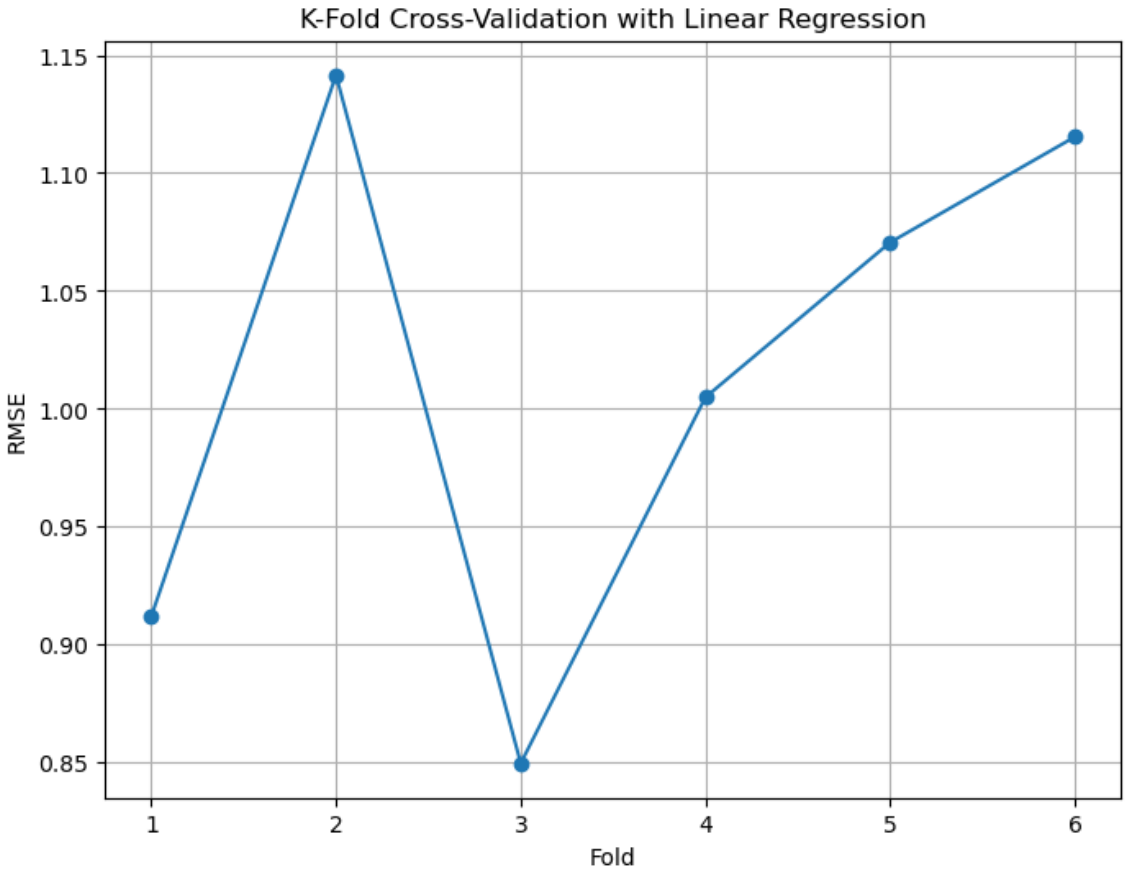
In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris  # You can replace this with you

# Load your dataset (replace with your data loading code)
data = load_iris()

# Split the dataset into features (X) and target labels (y)
X = data.data
y = data.target

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
X

# Now, you can train your machine learning model on X_train and y_train
# and test its performance on X_test and y_test.
```

Out[12]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

K-Fold Cross-Validation

In [37]:

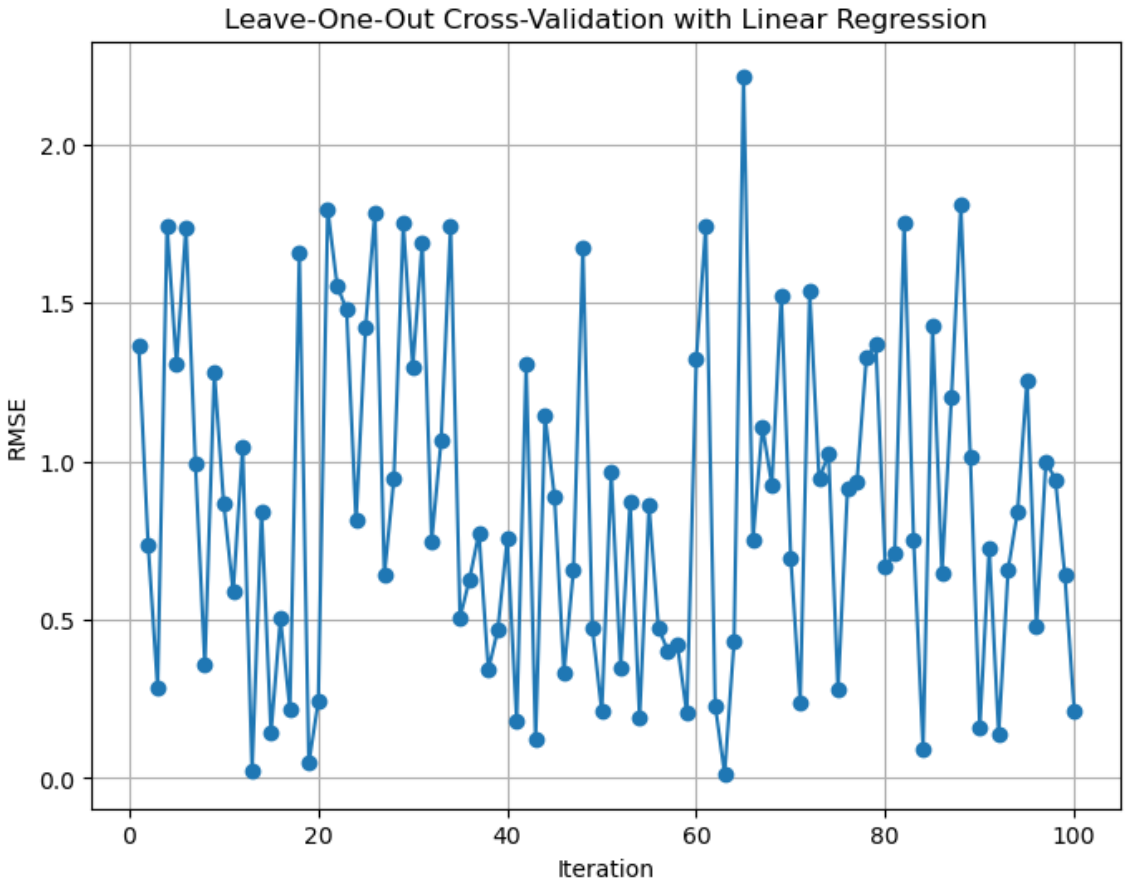
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 * X + 1 + np.random.randn(100, 1)

# Number of folds for cross-validation
n_folds = 6

# Initialize a KFold cross-validation splitter
kf = KFold(n_splits=n_folds, shuffle=True, random_state=0)

# Initialize an array to store RMSE for each fold
rmse_values = []

# Perform K-fold cross-validation
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Fit a linear regression model on the training data
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions on the test set
    predictions = model.predict(X_test)

    # Calculate RMSE for this fold
    rmse = np.sqrt(np.mean((predictions - y_test) ** 2))
    rmse_values.append(rmse)

# Plot the RMSE values for each fold
plt.figure(figsize=(8, 6))
plt.plot(range(1, n_folds + 1), rmse_values, marker='o', linestyle='-')
plt.xlabel('Fold')
plt.ylabel('RMSE')
plt.title('K-Fold Cross-Validation with Linear Regression')
plt.grid()
plt.show()
```



Leave-one-out cross validation

In [31]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import LeaveOneOut
from sklearn.linear_model import LinearRegression

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1)

# Initialize Leave-One-Out Cross-Validation
loo = LeaveOneOut()

# Initialize an array to store RMSE for each iteration
rmse_values = []

# Perform Leave-One-Out Cross-Validation
for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Fit a linear regression model on the training data
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Make predictions on the test data point
    prediction = model.predict(X_test)

    # Calculate RMSE for this iteration
    rmse = np.sqrt((prediction - y_test) ** 2)
    rmse_values.append(rmse)

# Flatten the RMSE values array
rmse_values = np.array(rmse_values).flatten()

# Plot the RMSE values for each iteration
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(rmse_values) + 1), rmse_values, marker='o', lines
plt.xlabel('Iteration')
plt.ylabel('RMSE')
plt.title('Leave-One-Out Cross-Validation with Linear Regression')
plt.grid()
plt.show()
# Calculate the average RMSE
average_rmse = np.mean(rmse_values)
print(f'Average RMSE: {average_rmse:.2f}')
```

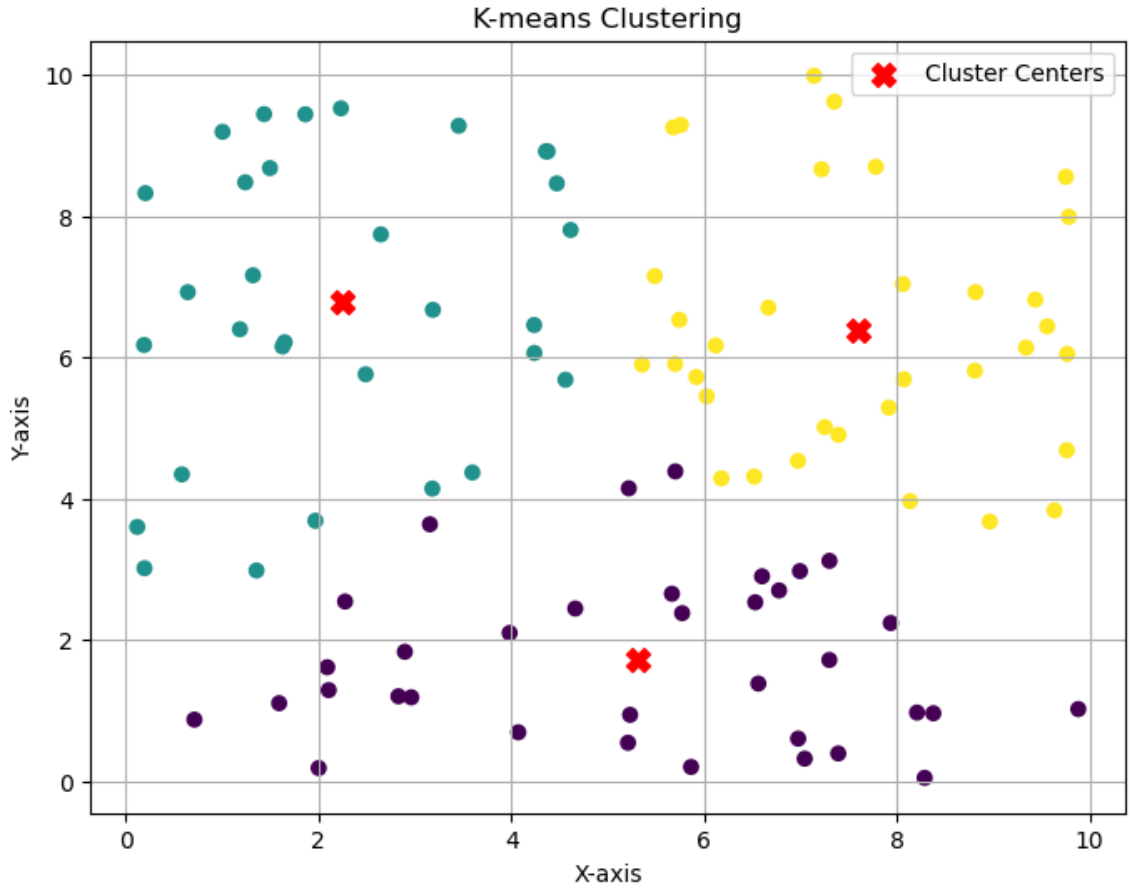



```
Average RMSE: 0.87
```

K-means

In [30]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 2) * 10

# Create a K-means model with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=0)

# Fit the K-means model to the data
kmeans.fit(X)

# Get cluster labels and cluster centers
cluster_labels = kmeans.labels_
cluster_centers = kmeans.cluster_centers_

# Plot the data points and cluster centers
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='red', mark
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('K-means Clustering')
plt.legend()
plt.grid()
plt.show()
```



R-SQUAR

In [39]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Generate synthetic data
np.random.seed(0)
X = np.random.rand(100, 1) * 10
y = 2 * X + 1 + np.random.randn(100, 1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = X[:80], X[80:], y[:80], y[80:]

# Fit a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)

# Calculate R-squared
r_squared = r2_score(y_test, predictions)

print(f'R-squared: {r_squared:.2f}')
```

```
R-squared: 0.97
```

In [4]:
```
from sklearn.metrics import r2_score

# Assuming you have observed actual and predicted values
actual_values = [10, 20, 30, 40, 50]
predicted_values = [12, 18, 28, 38, 52]

# Calculate the R-squared (R2) score
r2 = r2_score(actual_values, predicted_values)

print(f"R-squared (R2) Score: {r2}")

```

```
R-squared (R2) Score: 0.98
```

Mean absolute percentage error

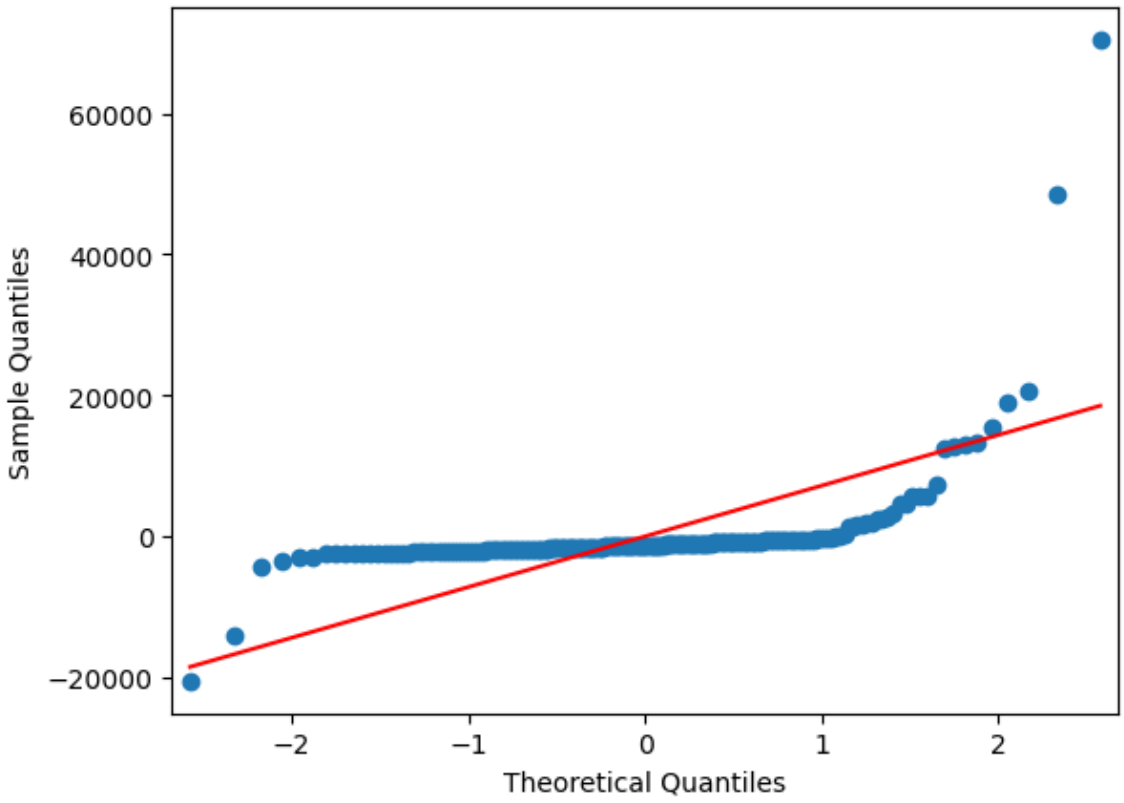
In [38]:
```
import numpy as np
absolute_percentage_errors = np.abs((y_test - y_pred) / y_test)
mape = np.mean(absolute_percentage_errors) * 100 # Multiply by 100 to e
mape
```

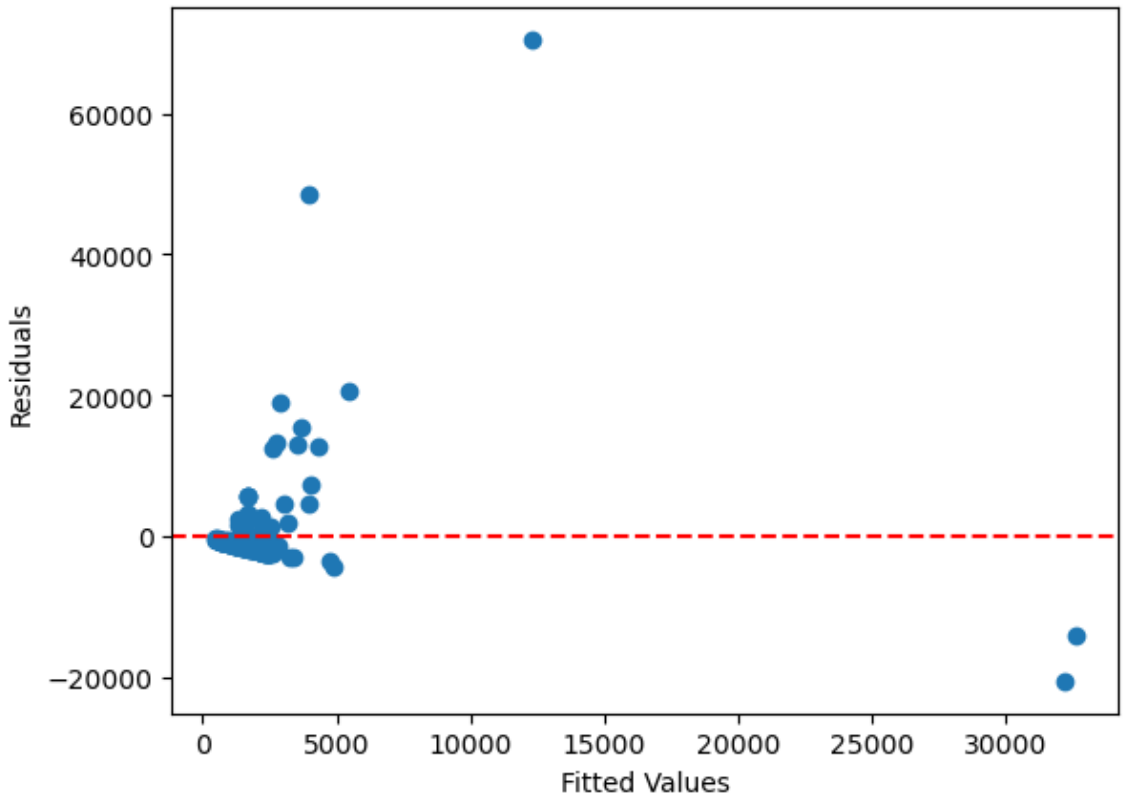
Out[38]:
```
5.555555555555338
```

Residual analysis

In [42]:

```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
df = pd.read_csv (r"C:\Users\Anusha V\Desktop\YouTube.csv")
X = df[['Likes', 'Rank']] # Features
y = df['Comments'] # Target variable
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
residuals = model.resid
sm.qqplot(residuals, line='s')
plt.show()
plt.scatter(model.fittedvalues, residuals)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.axhline(y=0, color='red', linestyle='--')
plt.show()

```





decision tree

In [2]:
```
# Import the necessary libraries
from sklearn import tree

# Create a dataset for training the decision tree
# In this example, we have two features (X) and a target variable (y)
X = [[0, 0], [1, 1]]
y = [0, 1]

# Create a decision tree classifier
clf = tree.DecisionTreeClassifier()

# Train the classifier on the dataset
clf = clf.fit(X, y)

# Make predictions using the trained decision tree
predictions = clf.predict([[2, 2], [0.5, 0.5]])

# Print the predictions
print(predictions)

```

```
[1 0]
```

Decision Tree Classifier

In [3]:

```
# Import the necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load a sample dataset (Iris dataset as an example)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Decision Tree Classifier
clf = DecisionTreeClassifier()

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

```
Accuracy: 100.00%
```

Binary Classifier

In [15]:

```
# Import the necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Generate a sample binary classification dataset (you can replace this
# Here, we'll create a simple dataset with two features (X) and binary
X = [[1.2, 2.3], [2.4, 3.5], [3.6, 4.7], [4.8, 5.9], [5.0, 6.1]]
y = [0, 1, 0, 1, 1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Binary Classifier (Logistic Regression is used in this examp
clf = LogisticRegression()

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

```
Accuracy: 0.00%
```

multi class classifier

Support Vector Machine (SVM)

In [17]:
```
# Import the necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load a sample dataset (Iris dataset as an example)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Support Vector Machine (SVM) classifier
clf = SVC(kernel='linear', C=1)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict
y_pred
```

Out[17]:
```
<bound method BaseSVC.predict of SVC(C=1, kernel='linear')>
```

In [18]:
```
# Import the necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load a sample dataset (Iris dataset as an example)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Support Vector Machine (SVM) classifier
clf = SVC(kernel='linear', C=1.0)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

```
Accuracy: 100.00%
```

multi-class classification

In [19]:

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load a sample dataset (Iris dataset as an example)
iris = load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate and print the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Print classification report (precision, recall, F1-score for each cla
report = classification_report(y_test, y_pred, target_names=iris.target
print(report)

```

```
Accuracy: 100.00%
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

multi-class classification

In [22]:

```
# Import the necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load a sample dataset (Iris dataset as an example)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Support Vector Machine (SVM) classifier
clf = SVC(kernel='linear', C=1.0)

# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

```
Accuracy: 100.00%
```

Confusion Matrix:

In [27]:

```python
import numpy as np

# Given confusion matrix
confusion_matrix = np.array([
    [13, 45],
    [0, 15]
])

# (i) Accuracy
total_samples = np.sum(confusion_matrix)
correct_predictions = np.trace(confusion_matrix)
accuracy = correct_predictions / total_samples
print(f"(i) Accuracy: {accuracy:.2f}")

# (ii) Precision for Class 1
precision_class_1 = confusion_matrix[1, 1] / (confusion_matrix[0, 1] +
print(f"(ii) Precision for Class 1: {precision_class_1:.2f}")

# (iii) Recall for Class 1
recall_class_1 = confusion_matrix[1, 1] / (confusion_matrix[1, 0] + con
print(f"(iii) Recall for Class 1: {recall_class_1:.2f}")

# (iv) Specificity for Class 0
specificity_class_0 = confusion_matrix[0, 0] / (confusion_matrix[0, 0]
print(f"(iv) Specificity for Class 0: {specificity_class_0:.2f}")

# (v) F1-Score for Class 1
f1_score_class_1 = 2 * (precision_class_1 * recall_class_1) / (precisio
print(f"(v) F1-Score for Class 1: {f1_score_class_1:.2f}")

```

```
(i) Accuracy: 0.38
(ii) Precision for Class 1: 0.25
(iii) Recall for Class 1: 1.00
(iv) Specificity for Class 0: 0.22
(v) F1-Score for Class 1: 0.40
```

In [22]:

```
import numpy as np

# Given confusion matrix
confusion_matrix = np.array([[0, 0, 0],
                             [13, 45, 8],
                             [0, 15, 32]])

# (i) Accuracy
total_samples = np.sum(confusion_matrix)
correct_predictions = np.trace(confusion_matrix)
accuracy = correct_predictions / total_samples
print(f"(i) Accuracy: {accuracy:.2f}")

# (ii) Precision for Class 1
precision_class_1 = confusion_matrix[1, 1] / (confusion_matrix[1, 0] +
print(f"(ii) Precision for Class 1: {precision_class_1:.2f}")

# (iii) Recall for Class 1
recall_class_1 = confusion_matrix[1, 1] / (confusion_matrix[0, 1] + con
print(f"(iii) Recall for Class 1: {recall_class_1:.2f}")

# (iv) Specificity for Class 0
specificity_class_0 = (confusion_matrix[0, 0] + confusion_matrix[2, 0])
print(f"(iv) Specificity for Class 0: {specificity_class_0:.2f}")


# (v) F1-Score for Class 1
f1_score_class_1 = 2 * (precision_class_1 * recall_class_1) / (precisio
print(f"(v) F1-Score for Class 1: {f1_score_class_1:.2f}")
```

```
(i) Accuracy: 0.68
(ii) Precision for Class 1: 0.68
(iii) Recall for Class 1: 0.75
(iv) Specificity for Class 0: 0.00
(v) F1-Score for Class 1: 0.71
```

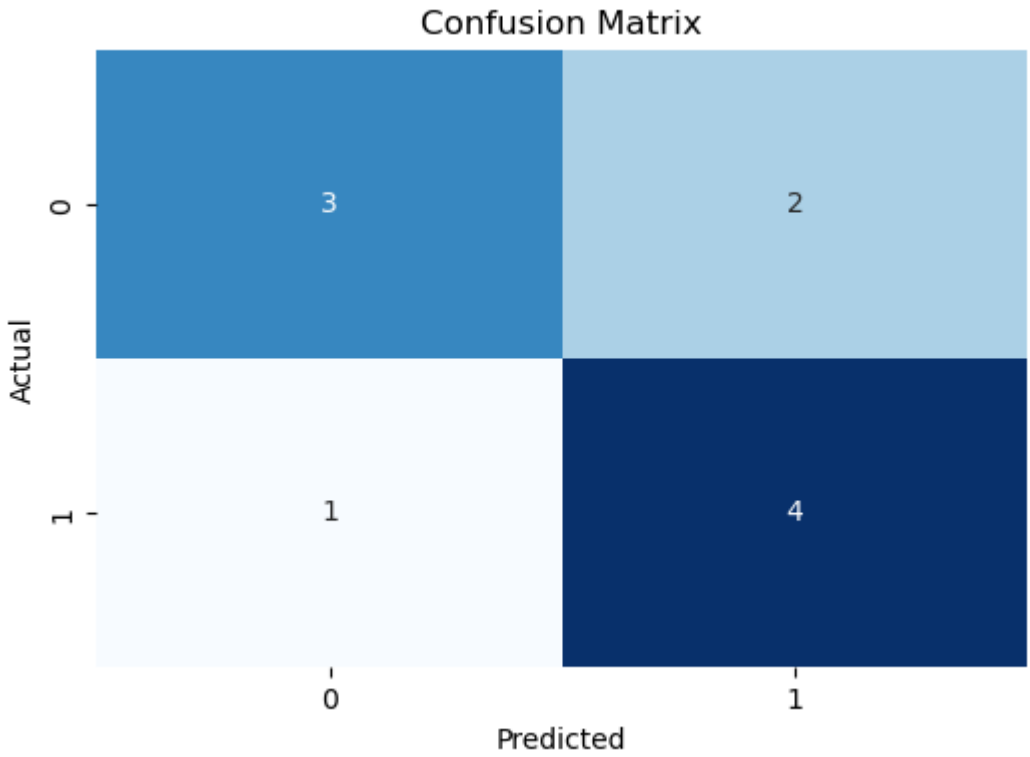
In [23]:

```
# Import the necessary libraries
from sklearn.metrics import confusion_matrix
import numpy as np

# Actual and predicted labels (replace these with your actual data)
actual = np.array([1, 0, 1, 1, 0, 1, 0, 1, 0, 0])
predicted = np.array([1, 0, 1, 0, 1, 1, 0, 1, 0, 1])

# Create the confusion matrix
cm = confusion_matrix(actual, predicted)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[3 2]
 [1 4]]
```

In [20]:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Example ground truth (actual) labels and predicted labels
true_labels = [1, 0, 1, 1, 0, 1, 0, 0, 1, 0]
predicted_labels = [1, 1, 0, 1, 0, 1, 0, 0, 1, 1]

# Create a confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)

# Display the confusion matrix as a heatmap
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

```



Creating a Confusion Matrix:

In [24]:
```
from sklearn.metrics import confusion_matrix

# True labels and predicted labels
true_labels = [0, 1, 1, 0, 2, 1, 2, 0]
predicted_labels = [0, 1, 0, 1, 2, 2, 2, 0]

# Generate the confusion matrix
cm = confusion_matrix(true_labels, predicted_labels)
print(cm)

```

```
[[2 1 0]
 [1 1 1]
 [0 0 2]]
```

Evaluating Performance Metrics:

In [27]:
```
from sklearn.metrics import classification_report

# Assuming 'true_labels' and 'predicted_labels' are defined
report = classification_report(true_labels, predicted_labels)
print(report)

```

```
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         3
           1       0.50      0.33      0.40         3
           2       0.67      1.00      0.80         2

    accuracy                           0.62         8
   macro avg       0.61      0.67      0.62         8
weighted avg       0.60      0.62      0.60         8
```

Custom Confusion Matrix Analysis:

In [31]:
```
tp = cm[1, 1]  # True Positives
fn = cm[1, 0]  # False Negatives
fp = cm[0, 1]  # False Positives
tn = cm[0, 0]  # True Negatives

sensitivity = tp / (tp + fn)  # Recall
specificity = tn / (tn + fp)
```

In [32]:
```
sensitivity
```

Out[32]: 0.5

In [33]:
```
specificity
```

Out[33]: 0.6666666666666666

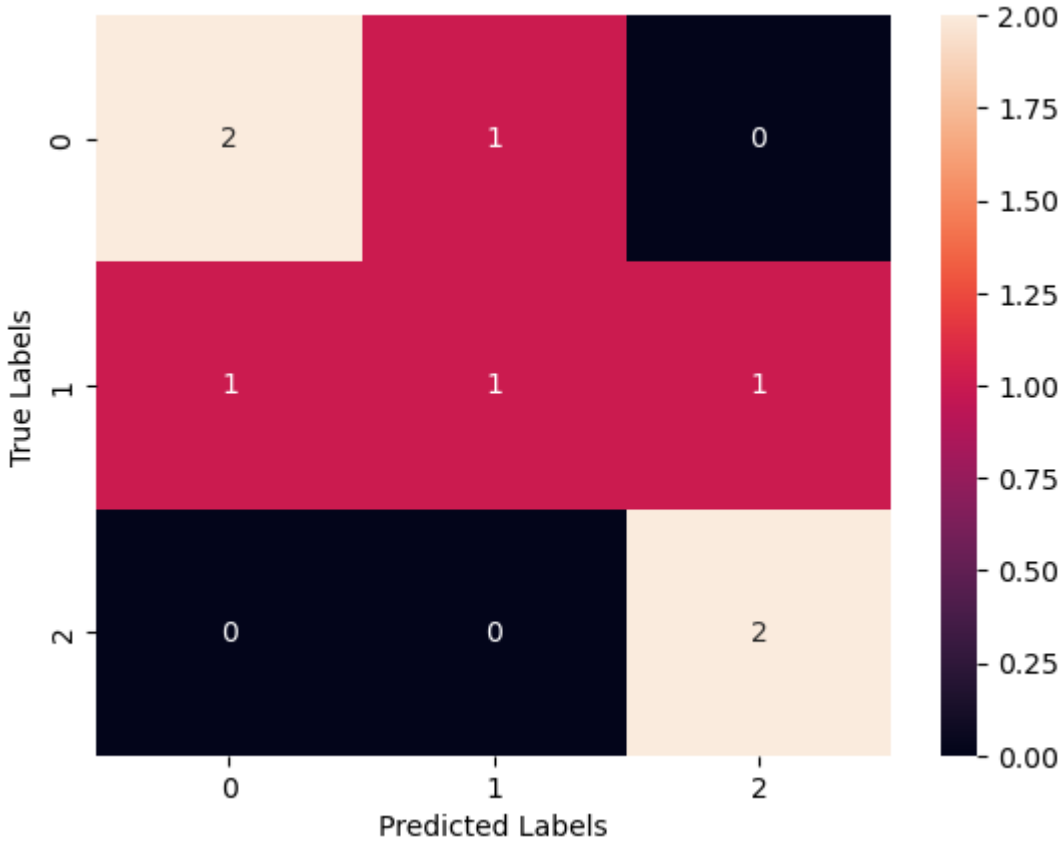
Multi-Class Confusion Matrix:

In [34]:
```
from sklearn.metrics import confusion_matrix

# Assuming 'true_labels' and 'predicted_labels' for multi-class
cm = confusion_matrix(true_labels, predicted_labels)
print(cm)

```

```
[[2 1 0]
 [1 1 1]
 [0 0 2]]
```

Visualizing a Confusion Matrix:

In [35]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'cm' is the confusion matrix from previous code
sns.heatmap(cm, annot=True, fmt="d")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

```



F1 score

In [50]:
```
from sklearn.metrics import f1_score

# Example true labels and predicted labels
true_labels = [1, 0, 1, 1, 0, 1, 0, 1]
predicted_labels = [1, 0, 0, 1, 0, 1, 1, 1]

# Calculate the F1 score
f1 = f1_score(true_labels, predicted_labels)
print(f"F1 Score: {f1:.2f}")

```

```
F1 Score: 0.80
```

DecisionTree Classifier hyperparameters

In [3]:
```
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris

# Load a sample dataset (you can replace this with your own dataset)
data = load_iris()
X, y = data.data, data.target

# Create a DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Define a dictionary of hyperparameters and their possible values
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create a GridSearchCV object to find the best hyperparameters
grid_search = GridSearchCV(clf, param_grid, cv=5, n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X, y)

# Print the best hyperparameters and the corresponding accuracy score
print("Best hyperparameters: ", grid_search.best_params_)
print("Best accuracy score: ", grid_search.best_score_)

```

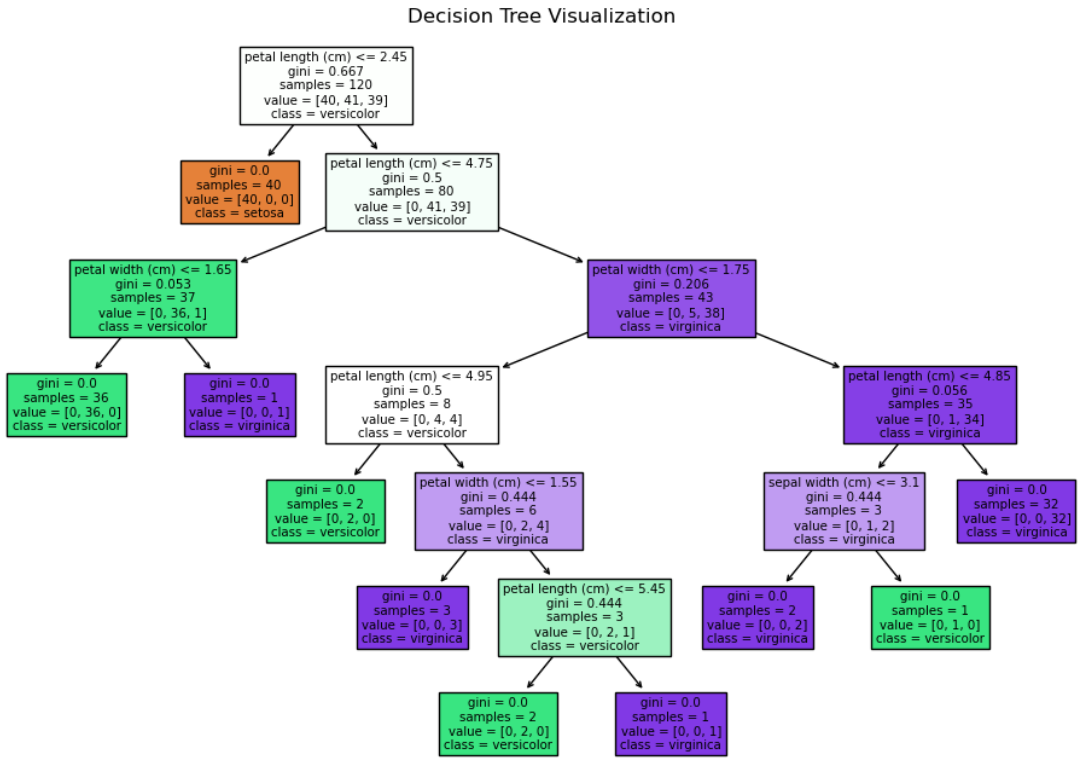
```
Best hyperparameters:  {'criterion': 'gini', 'max_depth': None, 'min_sampl
es_leaf': 1, 'min_samples_split': 5}
Best accuracy score:  0.9666666666666668
```

Decision Tree Visualization

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a Decision Tree classifier
clf = DecisionTreeClassifier(criterion='gini', max_depth=None, random_s

# Train the classifier
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

from sklearn.tree import plot_tree

# Plot the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=data.feature_names, class_nam
plt.title("Decision Tree Visualization")
plt.show()

```

```
Accuracy: 1.00
```



Loading [MathJax]/extensions/MathML/content-mathml.js

binary classification

In [42]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X = [[1.2, 2.3], [2.4, 3.5], [3.6, 4.7], [4.8, 5.9], [5.0, 6.1]]
y = [0, 1, 0, 1, 1]
# Load and preprocess your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create and train a binary classification model (e.g., logistic regres
model = LogisticRegression()
model.fit(X_train, y_train)


# Make predictions
predictions = model.predict(X_test)
predictions
```

Out[42]:
```
array([1])
```

In [39]:
```
# Import necessary libraries
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Create synthetic data for binary classification
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [2, 2], [3, 3], [4, 4], [
y = np.array([0, 0, 0, 0, 1, 1, 1, 1])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)



# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

predictions = model.predict(X_test)
predictions

```

```
Accuracy: 100.00%
```

Out[39]:
```
array([0, 1])
```

multi-class classification

In [8]:

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3

# Initialize the Support Vector Machine classifier
svm_classifier = SVC(kernel='linear', C=1)

# Fit the classifier to the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=iris.target
 /-
# Print the results
print("Accuracy:", accuracy)
print("Classification Report:\n", report)

```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

multi-label classification

In [48]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, hamming_loss

data = [
    ("This is a cat", {"animal": 1, "pet": 1}),
    ("This is a dog", {"animal": 1, "pet": 1}),
    ("This is a fish", {"animal": 1, "pet": 0}),
    ("This is a book", {"animal": 0, "pet": 0}),
    ("This is a rabbit", {"animal": 1, "pet": 1}),
]
# Split data into X (input) and y (labels)
X = [item[0] for item in data]
y = [item[1] for item in data]

# Convert labels to a binary matrix
mlb = MultiLabelBinarizer()
y_bin = mlb.fit_transform(y)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size

# Create a classifier (e.g., Support Vector Machine)
classifier = OneVsRestClassifier(SVC(kernel="linear"))

# Fit the model to the training data
classifier.fit(X_train, y_train)


# Predict labels for the test data
y_pred = classifier.predict(X_test)

# Inverse transform to get labels
y_pred_labels = mlb.inverse_transform(y_pred)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
hamming = hamming_loss(y_test, y_pred)

print("Accuracy: ", accuracy)
print("Hamming Loss: ", hamming)

```

```
Accuracy:  1.0
Hamming Loss:  0.0

C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\multiclass.py:79: Us
erWarning: Label 0 is present in all training examples.
  warnings.warn(
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\multiclass.py:79: Us
erWarning: Label 1 is present in all training examples.
  warnings.warn(
```

In [8]:
```
import sys
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import MultiLabelBinarizer

# Example data for training (adjust as needed)
X_train = ["text1", "text2", "text3"]
y_train = [["label1", "label2"], ["label2", "label3"], ["label1"]]

# Vectorize the text
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(X_train)

# Binarize the labels
mlb = MultiLabelBinarizer()
y_train = mlb.fit_transform(y_train)

# Train a multi-label classifier (e.g., SVM)
classifier = OneVsRestClassifier(SVC())
classifier.fit(X_train, y_train)


# Output the result
print("Predicted Labels: ", predicted_labels)

```

```
Predicted Labels:  classifier.fit
```

imbalanced classification

In [17]:
```
pip install -U imbalanced-learn

```

```
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
     ------------------------------------ 235.6/235.6 kB 720.1 kB/s eta 0:
00:00
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\anusha v\a
naconda3\lib\site-packages (from imbalanced-learn) (2.2.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\anusha v\anaconda
3\lib\site-packages (from imbalanced-learn) (1.21.5)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\anusha v\an
aconda3\lib\site-packages (from imbalanced-learn) (1.0.2)
Requirement already satisfied: scipy>=1.5.0 in c:\users\anusha v\anaconda3
\lib\site-packages (from imbalanced-learn) (1.9.1)
Collecting joblib>=1.1.1
  Downloading joblib-1.3.2-py3-none-any.whl (302 kB)
     ------------------------------------ 302.2/302.2 kB 1.3 MB/s eta 0:
00:00
Installing collected packages: joblib, imbalanced-learn
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.0
    Uninstalling joblib-1.1.0:
      Successfully uninstalled joblib-1.1.0
Successfully installed imbalanced-learn-0.11.0 joblib-1.3.2
Note: you may need to restart the kernel to use updated packages.
```

In [19]:
```
from imblearn.over_sampling import RandomOverSampler
from sklearn.ensemble import RandomForestClassifier

# Apply random oversampling to balance the data
oversampler = RandomOverSampler()
X_resampled, y_resampled = oversampler.fit_resample(X, y)

# Create a Random Forest classifier
model = RandomForestClassifier()

# Train the model on the resampled data
model.fit(X_resampled, y_resampled)

# Make predictions on new data
y_pred = model.predict(X_test)
y_pred
```

Out[19]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1])
```

Housing Price dataset

In [31]:

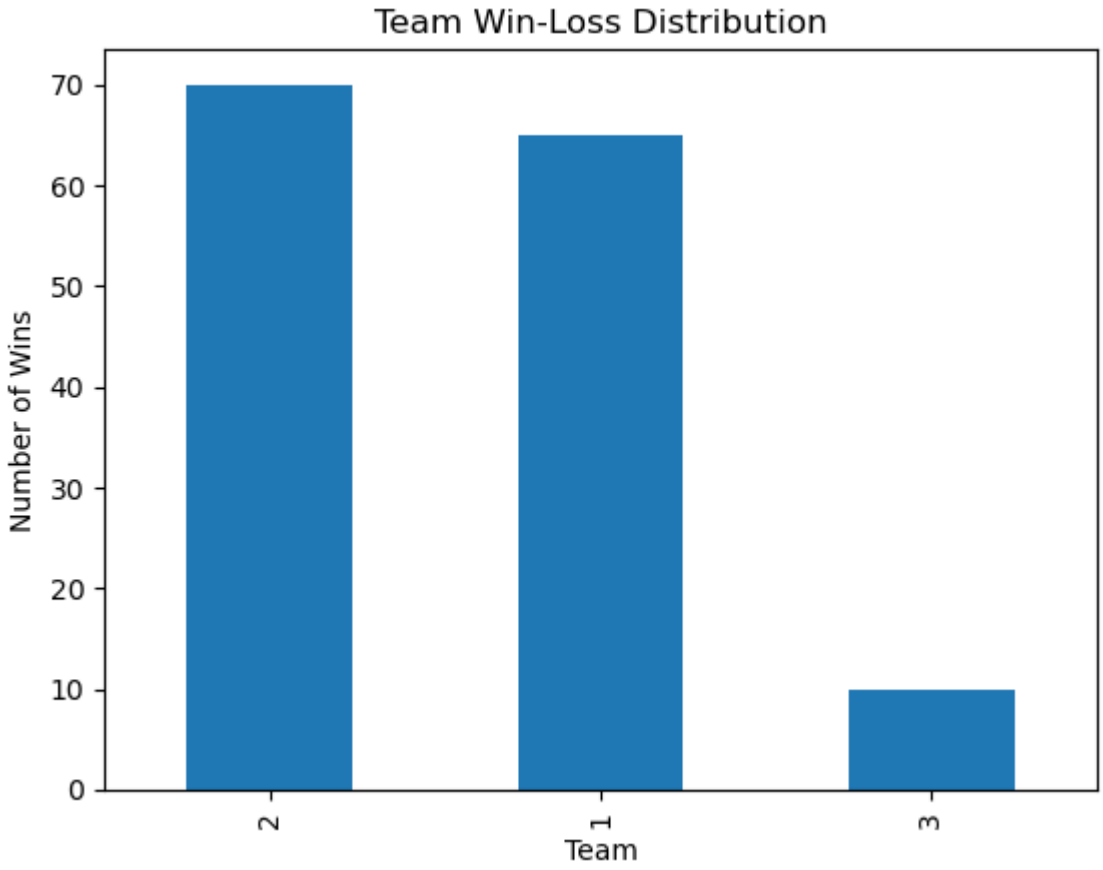
```
from sklearn.datasets import load_boston
import pandas as pd

# Load the Boston Housing Price dataset
boston = load_boston()

# Create a pandas DataFrame to explore the data
boston_df = pd.DataFrame(data=boston.data, columns=boston.feature_names
boston_df['target'] = boston.target

# Display the first few rows of the dataset
print(boston_df.head())

# Split the data into features (X) and target variable (y)
X = boston.data
y = boston.target

# You can use X and y for machine learning tasks such as regression:

# Example of training a linear regression model
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Split the data into a training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create and fit a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")

```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX  \
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0
2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0
3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0
4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0

   PTRATIO       B  LSTAT  target
0     15.3  396.90   4.98    24.0
1     17.8  396.90   9.14    21.6
2     17.8  392.83   4.03    34.7
3     18.7  394.63   2.94    33.4
4     18.7  396.90   5.33    36.2
Mean Squared Error: 24.29
R-squared (R2): 0.67
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\utils\deprecation.p
y:87: FutureWarning: Function load_boston is deprecated; `load_boston` is
deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refe
r to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of
this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=Non
e)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housi
ng
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

cricket match data

In [36]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load historical cricket match data
cricket_data = pd.read_csv(r"C:\Users\Anusha V\Downloads\DATA (1).csv")

# Perform EDA
# Example: Plot a bar chart to visualize the win-loss distribution of t
team_wins = cricket_data['wins'].value_counts()
team_wins.plot(kind='bar', title='Team Win-Loss Distribution')
plt.xlabel('Team')
plt.ylabel('Number of Wins')
plt.show()


```



In [ ]:
```
1
```

gini

In [6]:

```python
def gini_index(probabilities):
    """
    Calculate the Gini index for a set of class probabilities.

    :param probabilities: A list of class probabilities (e.g., [0.2, 0.
    :return: The Gini index for the given probabilities.
    """
    gini = 1.0  # Initialize the Gini index

    for p in probabilities:
        gini -= p ** 2

    return gini

# Example usage:
class_probabilities = [0.2, 0.3, 0.5]
gini = gini_index(class_probabilities)
print(f"Gini Index: {gini}")

```

```
Gini Index: 0.62
```

Entropy

In [1]:
```python
import math

def calculate_entropy(data):
    """
    Calculate the entropy of a set of data.

    Parameters:
    data (list): A list of labels or classes.

    Returns:
    float: The entropy of the data.
    """
    total_samples = len(data)
    class_counts = {}

    # Count the occurrences of each class in the data
    for label in data:
        if label in class_counts:
            class_counts[label] += 1
        else:
            class_counts[label] = 1

    entropy = 0.0

    # Calculate entropy using the formula: -p(x) * log2(p(x))
    for label in class_counts:
        probability = class_counts[label] / total_samples
        entropy -= probability * math.log2(probability)

    return entropy

# Example usage
data = ['A', 'B', 'A', 'C', 'B', 'A', 'A', 'B']
entropy = calculate_entropy(data)
print(f'Entropy of the data: {entropy}')

```

```
Entropy of the data: 1.4056390622295665
```

StandardScaler(Standardization)

In [25]:
```python
x = df.iloc[:, [7, 6]]
y = df.iloc[:, 5]
```

In [26]:
```python
x.head()
```

Out[26]:

	Comments	Likes
0	78	2700
1	18500	5300000
2	0	24700
3	9	166
4	0	12400

In [27]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [28]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state
```

In [29]:
```
x_train.head()
```

Out[29]:

	Comments	Likes
84	0	10800
16	7400	180300
19	4	1100
2	0	24700
142	108	7600

In [30]:
```
scaler = StandardScaler(). fit(x_train)
```

In [31]:
```
scaler
```

Out[31]:
```
StandardScaler()
```

In [32]:
```
scaler.mean_
```

Out[32]:
```
array([ 2221.26174497, 89513.84563758])
```

In [33]:
```
scaler.scale_
```

Out[33]:
```
array([  8663.65510725, 436331.85622113])
```

In [37]:
```
scaler.transform(x_train)
```

Out[37]:
```
array([[-2.56388524e-01, -1.80399035e-01],
       [ 5.97754434e-01,  2.08066757e-01],
       [-2.55926825e-01, -2.02629820e-01],
       [-2.56388524e-01, -1.48542548e-01],
       [-2.43922654e-01, -1.87732902e-01],
       [-2.49463041e-01, -2.01483904e-01],
       [-2.53964605e-01, -2.02941510e-01],
       [-2.53964605e-01, -2.03759694e-01],
       [-2.55349701e-01, -2.04621424e-01],
       [-2.55349701e-01, -2.04667260e-01],
       [-2.56388524e-01, -2.05150837e-01],
       [ 3.09192624e-01,  2.47027928e-01],
       [-2.56388524e-01, -1.78336384e-01],
       [-2.33303579e-01, -1.90712286e-01],
       [-2.56388524e-01, -1.99192070e-01],
       [-2.55349701e-01, -2.04596214e-01],
       [ 1.82225427e-01, -8.51151614e-03],
       [-2.15181897e-01, -1.42125414e-01],
       [-2.50155589e-01, -2.03335705e-01],
```

Loading [MathJax]/extensions/MathML/content-mathml.js

In [39]:
```
x_train_scales = scaler.transform(x_train)
x_train_scales
```

Out[39]:
```
array([[-2.56388524e-01, -1.80399035e-01],
       [ 5.97754434e-01,  2.08066757e-01],
       [-2.55926825e-01, -2.02629820e-01],
       [-2.56388524e-01, -1.48542548e-01],
       [-2.43922654e-01, -1.87732902e-01],
       [-2.49463041e-01, -2.01483904e-01],
       [-2.53964605e-01, -2.02941510e-01],
       [-2.53964605e-01, -2.03759694e-01],
       [-2.55349701e-01, -2.04621424e-01],
       [-2.55349701e-01, -2.04667260e-01],
       [-2.56388524e-01, -2.05150837e-01],
       [ 3.09192624e-01,  2.47027928e-01],
       [-2.56388524e-01, -1.78336384e-01],
       [-2.33303579e-01, -1.90712286e-01],
       [-2.56388524e-01, -1.99192070e-01],
       [-2.55349701e-01, -2.04596214e-01],
       [ 1.82225427e-01, -8.51151614e-03],
       [-2.15181897e-01, -1.42125414e-01],
       [-2.50155589e-01, -2.03335705e-01],
```

In [41]:
```
print(x_train_scales.mean(axis=0))
```

```
[ 0.0000000e+00 -2.9804645e-18]
```

In [43]:
```
print(x_train_scales.std(axis=0))
```

```
[1. 1.]
```

In [48]:
```
scaler = StandardScaler(). fit(x_test)
scaler.mean_
```

Out[48]:
```
array([  1724.14, 150251.96])
```

In [11]:
```
import pandas as pd
data = pd.read_csv (r"C:\Users\Anusha V\Downloads\archive (8)\APY.csv")
data.head(10)
```

Out[11]:

	State	District	Crop	Crop_Year	Season	Area	Production	Yield
0	Andaman and Nicobar Island	NICOBARS	Arecanut	2007	Kharif	2439.6	3415.0	1.40
1	Andaman and Nicobar Island	NICOBARS	Arecanut	2007	Rabi	1626.4	2277.0	1.40
2	Andaman and Nicobar Island	NICOBARS	Arecanut	2008	Autumn	4147.0	3060.0	0.74
3	Andaman and Nicobar Island	NICOBARS	Arecanut	2008	Summer	4147.0	2660.0	0.64
4	Andaman and Nicobar Island	NICOBARS	Arecanut	2009	Autumn	4153.0	3120.0	0.75
5	Andaman and Nicobar Island	NICOBARS	Arecanut	2009	Summer	4153.0	2080.0	0.50
6	Andaman and Nicobar Island	NICOBARS	Arecanut	2000	Kharif	1254.0	2000.0	1.59
7	Andaman and Nicobar Island	NICOBARS	Arecanut	2001	Kharif	1254.0	2061.0	1.64
8	Andaman and Nicobar Island	NICOBARS	Arecanut	2002	Whole Year	1258.0	2083.0	1.66
9	Andaman and Nicobar Island	NICOBARS	Arecanut	2003	Whole Year	1261.0	1525.0	1.21

In [14]:
```
X = data[['Crop_Year', 'Season', 'Yield']]
y = data['Crop_Year']
```

k mean

In [15]:
```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans

```

In [17]:
```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
```

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [19]:
```
k = 3 # Set the value of k
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train, y_train)
```

Out[19]: KNeighborsClassifier(n_neighbors=3)

In [20]:
```
y_pred = knn_classifier.predict(X_test)
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\neighbors\_classific
ation.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew
`, `kurtosis`), the default behavior of `mode` typically preserves the axi
s it acts along. In SciPy 1.11.0, this behavior will change: the default v
alue of `keepdims` will become False, the `axis` over which the statistic
is taken will be eliminated, and the value None will no longer be accepte
d. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [21]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

```

```
Accuracy: 100.00%
```

k MEAN

In [22]:
```
data=pd.read_csv(r"C:\Users\Anusha V\Downloads\archive (8)\APY.csv")
selected_columns = ['Crop_Year', 'Season', 'Yield']
X=data[selected_columns]

```

In [23]:
```
X.isnull().sum()
```

Out[23]:
```
Crop_Year    0
Season       0
Yield        0
dtype: int64
```

In [9]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import make_classification
```

In [10]:

```
import pandas as pd
from sklearn.datasets import make_classification

# Generate synthetic data
X, y = make_classification(n_features=5, n_redundant=0, n_informative=5

# Create a DataFrame
df = pd.DataFrame(X, columns=["Feature1", "Feature2", "Feature3", "Feat
df['target'] = y

# Print the shape and the first few rows of the DataFrame
print(df.shape)
print(df.head())

```

```
(100, 6)
   Feature1  Feature2  Feature3  Feature4  Feature5  target
0 -0.528476  2.049389  0.704279 -3.275012  3.555149       1
1  0.515171 -0.698543  0.194818 -3.008275 -0.890609       1
2  0.051224  1.454693 -1.665245 -1.872938  0.638409       1
3  1.920039  0.189161  1.949257 -0.355041 -2.558094       0
4  0.398833 -1.520716  0.397772 -0.063983  0.245394       0
```

In [15]:

```
import random

# function for row sampling
def sample_rows(df, percent):
    return df.sample(int(percent * df.shape[0]), replace=True)

# function for feature sampling
def sample_features(df, percent):
    cols = random.sample(df.columns.tolist()[:-1], int(percent * df.sha
    return df[cols]

# Example usage
# df is your DataFrame
# Sample 50% of rows
sampled_rows = sample_rows(df, 0.5)

# Sample 50% of features (columns)
sampled_features = sample_features(df, 0.5)
sampled_features


```

Out[15]:

	Feature3	Feature1	Feature5
0	0.704279	-0.528476	3.555149
1	0.194818	0.515171	-0.890609
2	-1.665245	0.051224	0.638409
3	1.949257	1.920039	-2.558094
4	0.397772	0.398833	0.245394
...	...	...	...
95	0.580032	-0.974712	0.495764
96	2.427803	-0.627004	3.220083
97	-0.059303	-0.836839	0.086734
98	1.429864	-0.874207	-1.519296
99	0.378503	0.544607	-2.403189

100 rows × 3 columns

In [22]:

```
def combined_sampling (df,row_percent, col_percent):
 new_df = sample_rows(df,row_percent)
 return sample_features (new_df,col_percent)
import random

# Example usage
# df is your DataFrame
# Sample 50% of rows and 50% of features (columns)
sampled_data = combined_sampling(df, 0.5, 0.5)
sampled_data
```

Out[22]:

	Feature5	Feature2	Feature1
25	2.383718	-0.194155	1.953884
31	0.162926	-0.245893	2.985782
11	1.199305	3.189073	1.504901
90	0.660209	-2.729023	-0.753172
9	-2.700445	2.972274	0.472425
81	-4.573964	2.196432	2.500614
89	-4.038451	2.387250	1.216692
37	1.463461	-1.402099	0.998925
67	2.544064	-0.610241	0.347334
54	-2.098093	1.744157	0.603254
77	2.639847	-0.982625	1.127300
80	-0.606813	-0.964081	1.735420
88	0.099062	-0.056112	1.762495
97	0.086734	-1.007120	-0.836839
51	0.581332	-0.986208	0.246330
97	0.086734	-1.007120	-0.836839
95	0.495764	-2.351240	-0.974712
78	-2.006040	1.333253	0.764933
95	0.495764	-2.351240	-0.974712
45	0.047452	-1.057637	1.744233
19	0.008011	-1.770500	2.099001
7	1.537535	-1.629851	0.325215
9	-2.700445	2.972274	0.472425
78	-2.006040	1.333253	0.764933
0	3.555149	2.049389	-0.528476
72	-1.738130	-1.777368	1.360138
26	0.646512	0.972441	2.132425
57	-1.258582	-1.667245	1.509598
96	3.220083	-2.182029	-0.627004
1	-0.890609	-0.698543	0.515171
4	0.245394	-1.520716	0.398833
25	2.383718	-0.194155	1.953884
1	-0.890609	-0.698543	0.515171
92	2.917935	2.023404	-1.276507
31	0.162926	-0.245893	2.985782
71	-0.543031	-0.412917	-2.981880
14	-0.654628	0.111394	-0.737606
51	0.581332	-0.986208	0.246330
80	-0.606813	-0.964081	1.735420

	Feature5	Feature2	Feature1
9	-2.700445	2.972274	0.472425
27	1.832087	-1.387378	2.833763
52	3.108148	1.295792	1.440234
2	0.638409	1.454693	0.051224
24	2.737663	-2.721685	2.150645
51	0.581332	-0.986208	0.246330
26	0.646512	0.972441	2.132425
65	-2.930517	-0.468903	3.043369
72	-1.738130	-1.777368	1.360138
49	-0.716775	-1.915392	-2.627376
78	-2.006040	1.333253	0.764933

In [23]:
```
df1 = sample_rows(df,0.1)
df1.shape

```

Out[23]: (10, 6)

In [24]:
```
df1 = sample_rows(df,0.1)
df1.shape
```

Out[24]: (10, 6)

In [25]:
```
df3 = sample_rows(df,0.1)
df3.shape

```

Out[25]: (10, 6)

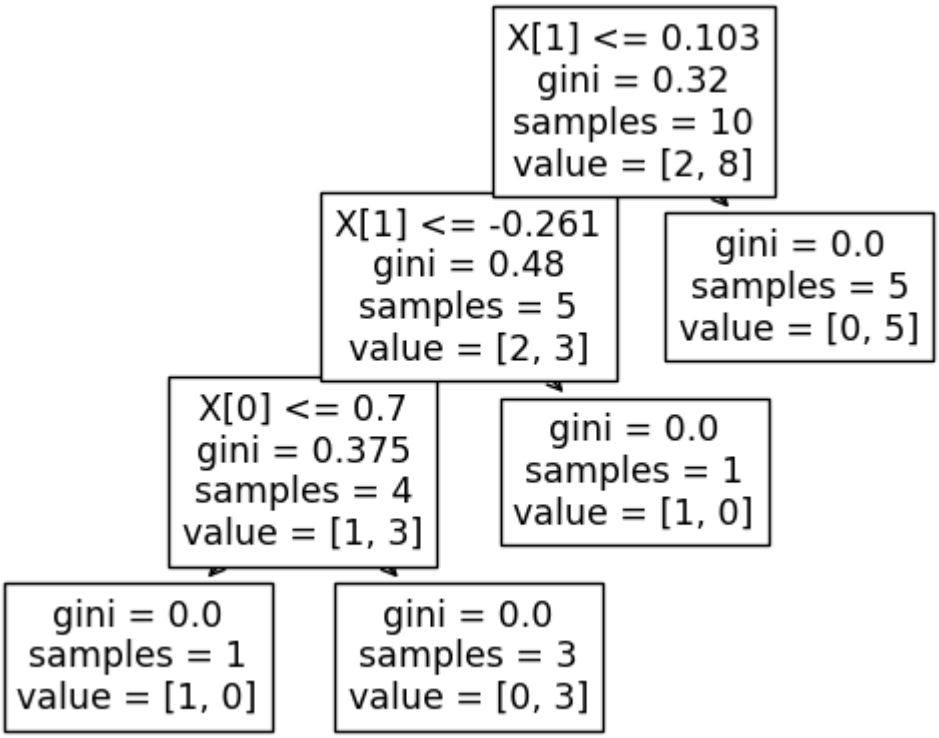
In [26]:
```
from sklearn.tree import DecisionTreeClassifier
clf1 = DecisionTreeClassifier()
c1f2 = DecisionTreeClassifier()
c1f3 = DecisionTreeClassifier()

```

In [27]:
```
clf1.fit(df1.iloc[:,0:2],df1.iloc[:,-1])
c1f2.fit(df2.iloc[:,0:2],df2.iloc[:,-1])
c1f3.fit(df3.iloc[:,0:2],df3.iloc[:,-1])
```

Out[27]: DecisionTreeClassifier()

In [28]:
```
1 from sklearn.tree import plot_tree
2 plot_tree(clf1)
```

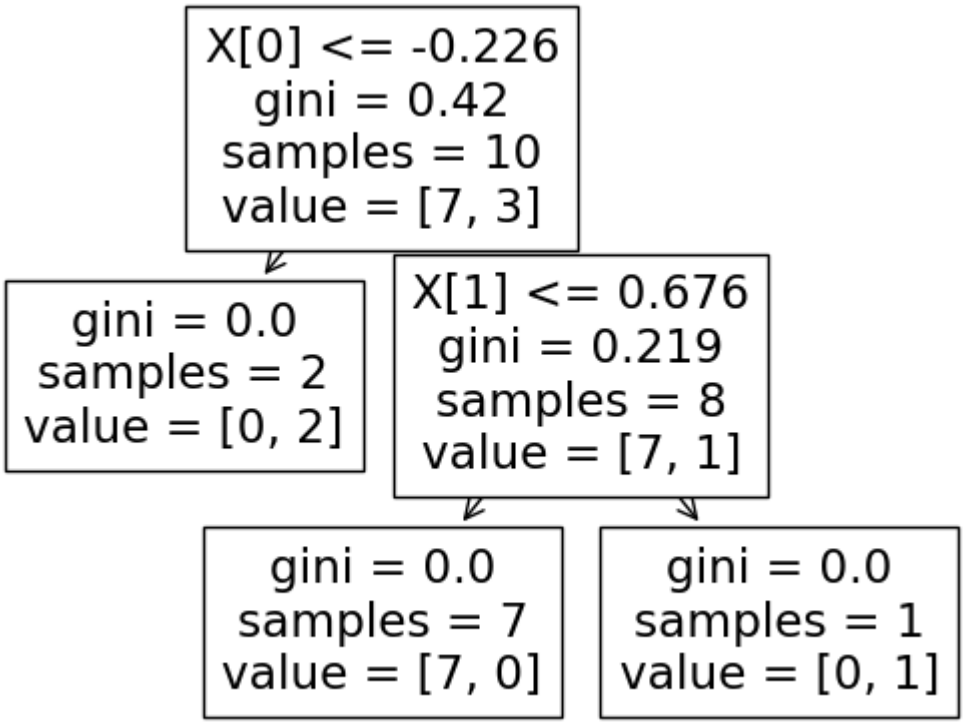
Out[28]:
```
[Text(0.6666666666666666, 0.875, 'X[1] <= 0.103\ngini = 0.32\nsamples = 10
\nvalue = [2, 8]'),
 Text(0.5, 0.625, 'X[1] <= -0.261\ngini = 0.48\nsamples = 5\nvalue = [2,
3]'),
 Text(0.3333333333333333, 0.375, 'X[0] <= 0.7\ngini = 0.375\nsamples = 4\n
value = [1, 3]'),
 Text(0.16666666666666666, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.5, 0.125, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.6666666666666666, 0.375, 'gini = 0.0\nsamples = 1\nvalue = [1,
0]'),
 Text(0.8333333333333334, 0.625, 'gini = 0.0\nsamples = 5\nvalue = [0,
5]')]
```



Loading [MathJax]/extensions/MathML/content-mathml.js

In [29]:
```
1 plot_tree(c1f2)
```

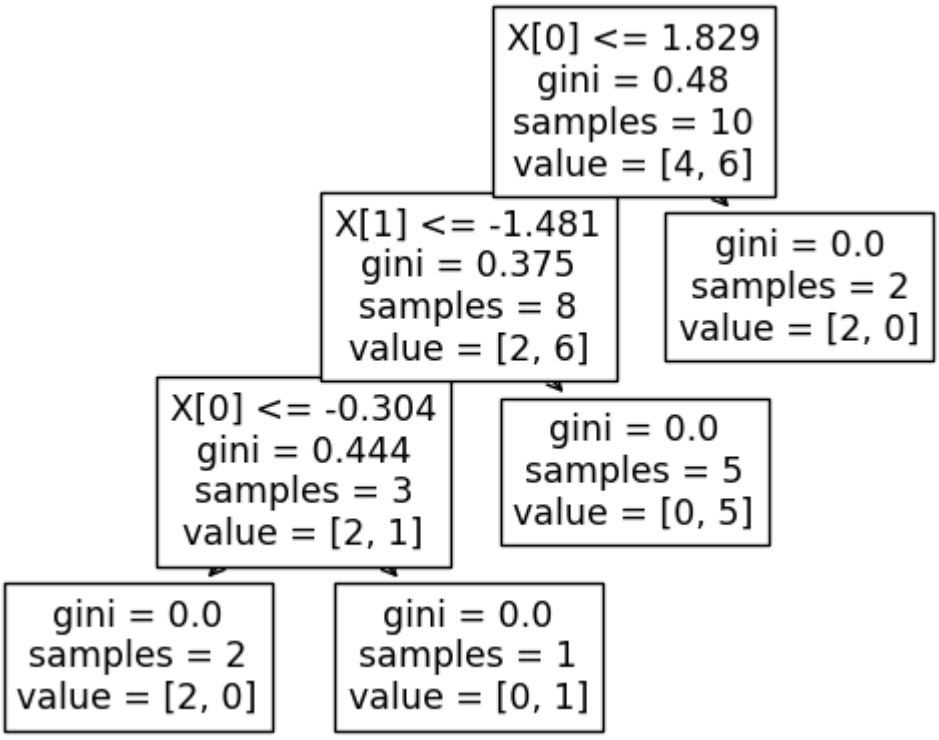
Out[29]:
```
[Text(0.4, 0.8333333333333334, 'X[0] <= -0.226\ngini = 0.42\nsamples = 10
\nvalue = [7, 3]'),
 Text(0.2, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.6, 0.5, 'X[1] <= 0.676\ngini = 0.219\nsamples = 8\nvalue = [7,
1]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.0\nsamples = 7\nvalue = [7,
0]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]')]
```



Loading [MathJax]/extensions/MathML/content-mathml.js

In [30]:
```
1 plot_tree(c1f3)
```

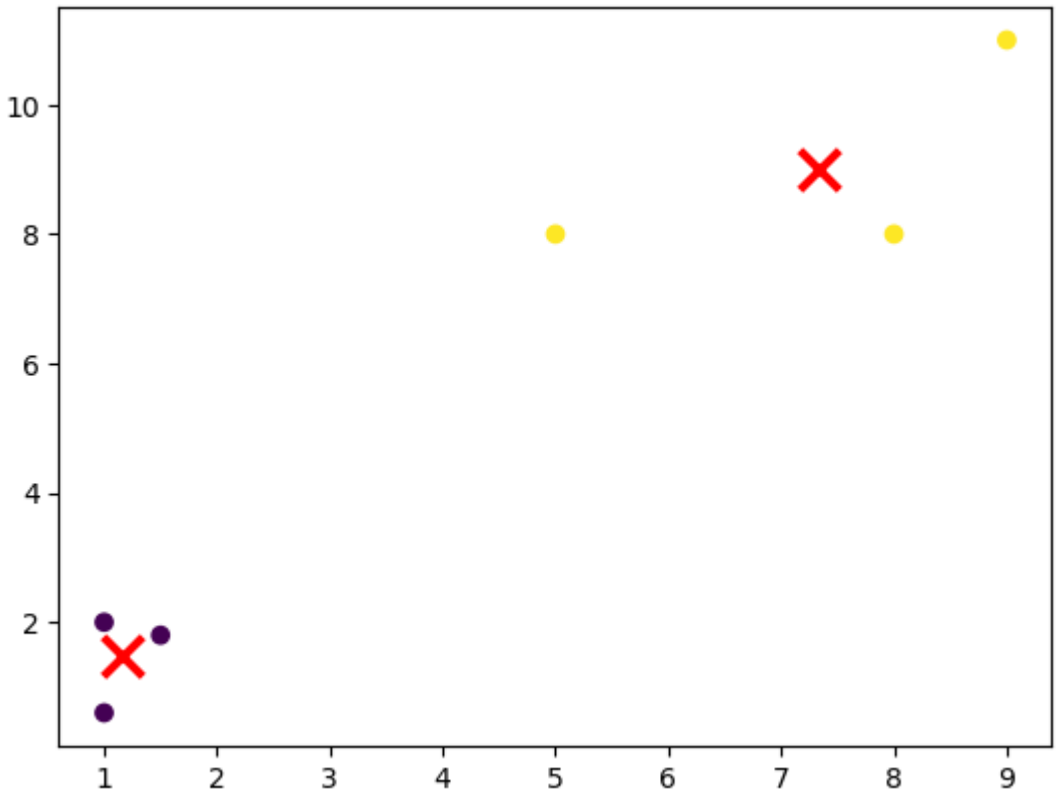
Out[30]:
```
[Text(0.6666666666666666, 0.875, 'X[0] <= 1.829\ngini = 0.48\nsamples = 10
\nvalue = [4, 6]'),
 Text(0.5, 0.625, 'X[1] <= -1.481\ngini = 0.375\nsamples = 8\nvalue = [2,
6]'),
 Text(0.3333333333333333, 0.375, 'X[0] <= -0.304\ngini = 0.444\nsamples =
3\nvalue = [2, 1]'),
 Text(0.16666666666666666, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.5, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.6666666666666666, 0.375, 'gini = 0.0\nsamples = 5\nvalue = [0,
5]'),
 Text(0.8333333333333334, 0.625, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]')]
```



Loading [MathJax]/extensions/MathML/content-mathml.js

In [31]:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Generate some example data (replace this with your own data)
import numpy as np
X = np.array([[1, 2], [5, 8], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]])

# Create a K-Means model with the desired number of clusters (e.g., 2)
kmeans = KMeans(n_clusters=2)

# Fit the model to your data
kmeans.fit(X)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Get the coordinates of the cluster centers
centers = kmeans.cluster_centers_

# Plot the data points and cluster centers
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], marker='x', s=200, linewidths
plt.show()

```



hierarchical clustering

In [33]:
```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=3)
labels = clustering.fit_predict(X)
labels
```

Out[33]:
```
array([1, 0, 1, 0, 1, 2], dtype=int64)
```

density based clustering

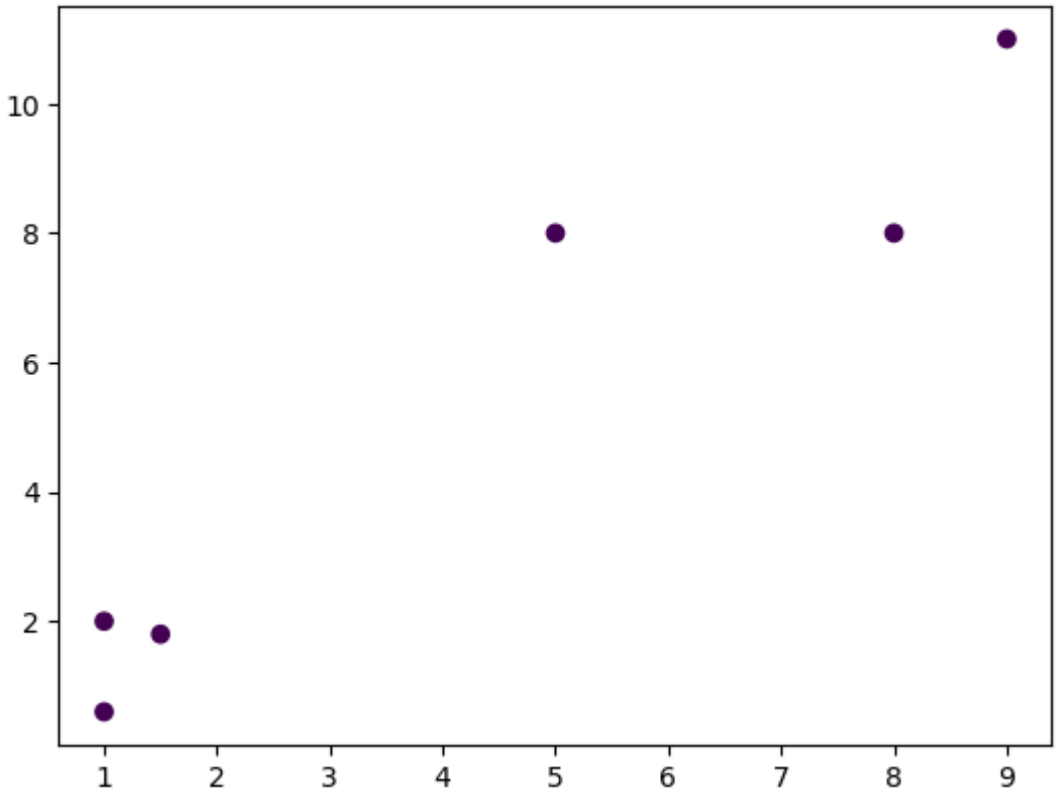
In [34]:
```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(X)
labels
```

Out[34]:
```
array([-1, -1, -1, -1, -1, -1], dtype=int64)
```

partitional clustering

In [36]:

```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# Generate some example data (replace this with your own data)
import numpy as np
X = np.array([[1, 2], [5, 8], [1.5, 1.8], [8, 8], [1, 0.6], [9, 11]])

# Create a DBSCAN model
dbscan = DBSCAN(eps=0.3, min_samples=2)

# Fit the model to your data
dbscan.fit(X)

# Get the cluster labels for each data point (-1 represents noise)
labels = dbscan.labels_

# Plot the data points and clusters (including noise)
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.show()

```



distribution model based of cluster

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Generate some example data (replace this with your own data)
data = np.random.randn(300, 2)
data[100:200, 0] += 5
data[200:300, 1] += 5

# Create a K-Means model with the desired number of clusters (e.g., 3)
kmeans = KMeans(n_clusters=3)
kmeans

```

Out[37]:
```
KMeans(n_clusters=3)
```

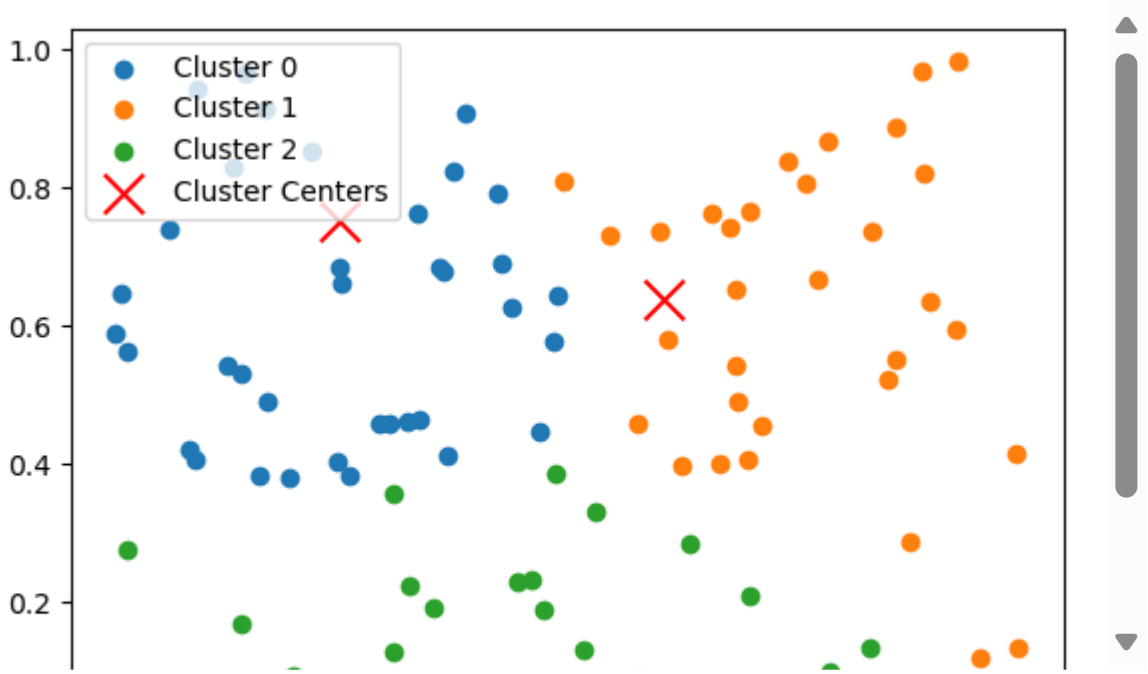
fuzzy clustering

In [39]:
```
pip install -U scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
     ------------------------------------ 994.0/994.0 kB 31.2 kB/s eta 0:
00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: numpy>=1.6.0 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-fuzzy) (1.21.5)
Requirement already satisfied: scipy>=0.9.0 in c:\users\anusha v\anaconda3
\lib\site-packages (from scikit-fuzzy) (1.9.1)
Requirement already satisfied: networkx>=1.9.0 in c:\users\anusha v\anacon
da3\lib\site-packages (from scikit-fuzzy) (2.8.4)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py): started
  Building wheel for scikit-fuzzy (setup.py): finished with status 'done'
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-an
y.whl size=894075 sha256=4343413b7cb37e9505f40ff3362f46f146bfd963813d0a4de
12125836171b977
  Stored in directory: c:\users\anusha v\appdata\local\pip\cache\wheels\32
\2c\a1\a90a7d7dd8448ec029f298a61f3490275e99b17aa348be675c
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
Note: you may need to restart the kernel to use updated packages.

WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'ReadTimeoutError("HTTPSConnectio
nPool(host='pypi.org', port=443): Read timed out. (read timeout=15)")': /s
imple/scikit-fuzzy/
WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'ReadTimeoutError("HTTPSConnectio
nPool(host='pypi.org', port=443): Read timed out. (read timeout=15)")': /s
imple/scikit-fuzzy/
WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None,
status=None)) after connection broken by 'ReadTimeoutError("HTTPSConnectio
nPool(host='pypi.org', port=443): Read timed out. (read timeout=15)")': /s
imple/scikit-fuzzy/
```

In [40]:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy.cluster import cmeans
import matplotlib.pyplot as plt

# Generate some example data (replace this with your own data)
data = np.random.rand(2, 100)

# Specify the number of clusters
n_clusters = 3

# Specify the fuzziness exponent (typically 2.0 for FCM)
m = 2.0

# FCM clustering
cntr, u, u0, d, jm, p, fpc = cmeans(data, n_clusters, m, error=0.005, m

# Get the cluster memberships for each data point
cluster_membership = np.argmax(u, axis=0)

# Visualize the data points and cluster centers
for i in range(n_clusters):
    plt.scatter(data[0, cluster_membership == i], data[1, cluster_membe

plt.scatter(cntr[0], cntr[1], s=200, c='red', marker='x', label='Cluste
plt.legend()
plt.show()

```



knn classification

Loading [MathJax]/extensions/MathML/content-mathml.js

In [2]:

```
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load a sample dataset (Iris dataset)
iris = load_iris()
X, y = iris.data, iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Create a k-NN classifier
knn = KNeighborsClassifier(n_neighbors=3)

# Train the classifier on the training data
knn.fit(X_train, y_train)

# Predict the class labels for the test data
y_pred = knn.predict(X_test)
y_pred
```

```
C:\Users\Anusha V\anaconda3\lib\site-packages\sklearn\neighbors\_classific
ation.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew
`, `kurtosis`), the default behavior of `mode` typically preserves the axi
s it acts along. In SciPy 1.11.0, this behavior will change: the default v
alue of `keepdims` will become False, the `axis` over which the statistic
is taken will be eliminated, and the value None will no longer be accepte
d. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Out[2]:

```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

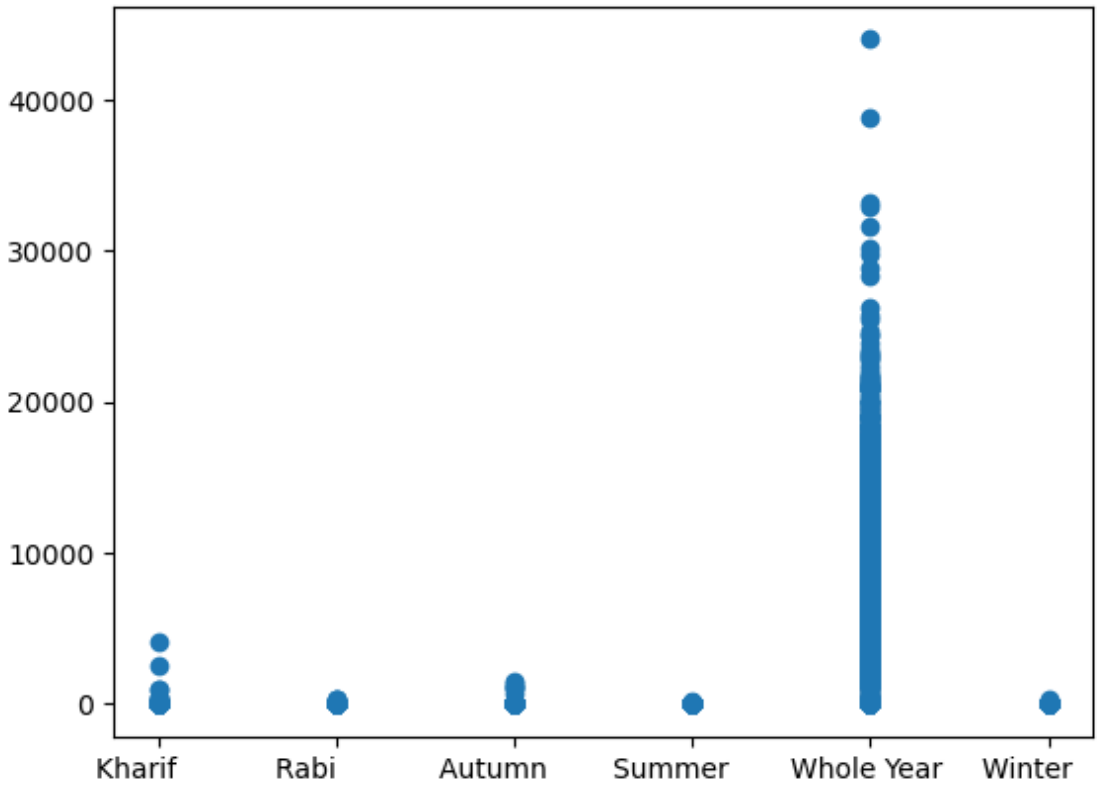
In [18]:
```
df=pd.read_csv(r"C:\Users\Anusha V\Downloads\archive (8)\APY.csv")
df.head(10)
```

Out[18]:

	State	District	Crop	Crop_Year	Season	Area	Production	Yield
0	Andaman and Nicobar Island	NICOBARS	Arecanut	2007	Kharif	2439.6	3415.0	1.40
1	Andaman and Nicobar Island	NICOBARS	Arecanut	2007	Rabi	1626.4	2277.0	1.40
2	Andaman and Nicobar Island	NICOBARS	Arecanut	2008	Autumn	4147.0	3060.0	0.74
3	Andaman and Nicobar Island	NICOBARS	Arecanut	2008	Summer	4147.0	2660.0	0.64
4	Andaman and Nicobar Island	NICOBARS	Arecanut	2009	Autumn	4153.0	3120.0	0.75
5	Andaman and Nicobar Island	NICOBARS	Arecanut	2009	Summer	4153.0	2080.0	0.50
6	Andaman and Nicobar Island	NICOBARS	Arecanut	2000	Kharif	1254.0	2000.0	1.59
7	Andaman and Nicobar Island	NICOBARS	Arecanut	2001	Kharif	1254.0	2061.0	1.64
8	Andaman and Nicobar Island	NICOBARS	Arecanut	2002	Whole Year	1258.0	2083.0	1.66
9	Andaman and Nicobar Island	NICOBARS	Arecanut	2003	Whole Year	1261.0	1525.0	1.21

In [19]:
```
import matplotlib.pyplot as plt
plt.scatter(df["Season"],df["Yield"])
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x2126a43d8e0>
```



In [23]:
```
from sklearn.cluster import KMeans
import numpy as np

# Sample data
data = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])

# Create a KMeans object with the desired number of clusters
km = KMeans(n_clusters=3)

# Fit the data and predict the cluster labels
cluster_labels = km.fit_predict(data)

# The cluster_labels now contain the assigned cluster labels for each d
print(cluster_labels)

```

```
[0 1 0 2 1 2]
```

In [ ]:
```

```

Breast Cancer Wisconsin (diagnostic)

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [4]:
```
data = load_breast_cancer()
X = data.data
y = data.target
```

In [6]:
```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=4
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [7]:
```
rf_classifier. fit(X_train, y_train)

```

Out[7]:
```
RandomForestClassifier(random_state=42)
```

In [9]:
```
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Out[9]:
```
0.9649122807017544
```

In [10]:
```
report = classification_report(y_test, y_pred)
print(report)

```

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Principal Component Analysis (PCA)

In [17]:
```
from sklearn.decomposition import PCA
import numpy as np

# Sample data
data = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Create a PCA object with the desired number of components
pca = PCA(n_components=2)

# Fit the data and transform it to the reduced dimension
transformed_data = pca.fit_transform(data)

# The transformed_data now contains the reduced dimension representatio
print(transformed_data)

```

```
[[ 5.19615242  0.        ]
 [-0.          0.        ]
 [-5.19615242  0.        ]]
```

In [30]:
```
import pandas as pd
import numpy as np
import random as rd
from sklearn.decomposition import PCA
from sklearn import preprocessing
import matplotlib.pyplot as plt
```

In [6]:
```
import sys
import pandas as pd
from sklearn.decomposition import PCA


```

In [ ]:
```
import sys
import pandas as pd
from sklearn.decomposition import PCA

# Extract the user's input from the command line arguments (e.g., the d
data_filename = sys.argv[1]

# Load the data from the CSV file
data = pd.read_csv(data_filename)

# Perform PCA on the data
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data)

# Process the PCA result as needed and return it
print("PCA Result:")
print(pca_result)

```

In [7]:
```
data_filename = sys.argv[1]

```

In [14]:
```
data = pd.read_csv(r"C:\Users\Anusha V\Desktop\YouTube.csv")


```

In [16]:
```
from sklearn.decomposition import PCA

# Assuming 'data' is your data in the form of a NumPy array or a Pandas
pca = PCA(n_components=2)
pca_result = pca


```

In [17]:
```
print("PCA Result:")
print(pca_result)

```

```
PCA Result:
PCA(n_components=2)
```

In [19]:
```
import pandas as pd
data = pd.read_csv("C:/Users/Anusha V/Downloads/archive (12)/diabetes.c
data.head()
```

Out[19]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.(
1	1	85	66	29	0	26.6	0.:
2	8	183	64	0	0	23.3	0.(
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.:

In [21]:
```
1 data.isnull().sum()
```

Out[21]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [22]:
```
1 data.describe()
```

Out[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diab
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [23]:
```
1 data.Outcome.value_counts()
```

Out[23]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [24]:
```
1 268/500
```

Out[24]:
```
0.536
```

In [25]:
```
1 x = data.drop('Outcome', axis='columns')
```

In [26]:
```
1 y = data.Outcome
```

In [27]:
```
1 from sklearn.preprocessing import StandardScaler
```

In [28]:
```
1 scaler = StandardScaler()
```

In [33]:
```
X_scaled = scaler.fit_transform(x)
X_scaled[:3]
```

Out[33]:
```
array([[ 0.63994726,  0.84832379,  0.14964075,  0.90726993, -0.69289057,
         0.20401277,  0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575,  0.53090156, -0.69289057,
        -0.68442195, -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, -1.28821221, -0.69289057,
        -1.10325546,  0.60439732, -0.10558415]])
```

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
X_train,X_test,y_train,y_test = train_test_split(X_scaled,y, stratify=y
```

In [38]:
```
X_train.shape
```

Out[38]:
```
(576, 8)
```

In [39]:
```
X_test.shape
```

Out[39]:
```
(192, 8)
```

In [41]:
```
y_train.value_counts
```

Out[41]:
```
<bound method IndexOpsMixin.value_counts of 745    0
67     0
570    0
517    0
717    0
      ..
735    0
475    0
46     0
767    0
540    1
Name: Outcome, Length: 576, dtype: int64>
```

In [42]:
```
201/375
```

Out[42]:
```
0.536
```

In [43]:
```
451/854
```

Out[43]:
```
0.5281030444964872
```

In [44]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [45]:
```
from sklearn.model_selection import cross_val_score
```

In [49]:
```
scores = cross_val_score(DecisionTreeClassifier(), x, y, cv=5)
scores
```

Out[49]:
```
array([0.66233766, 0.68181818, 0.68831169, 0.79084967, 0.74509804])
```

In [50]:
```
scores.mean()
```

Out[50]:
```
0.7136830489771666
```

In [57]:
```
from sklearn.ensemble import BaggingClassifier
BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    n_estimators=00,
    max_samples=0.8,
    oob_score=True,
    random_state=0
)


```

Out[57]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(), max_samples=0.
8,
                  n_estimators=0, oob_score=True, random_state=0)
```

In [2]:
```
import pandas as pd
df = pd.read_csv(r"C:\Users\Anusha V\Downloads\archive (13)\healthcare-
df
```

Out[2]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_
0	9046	Male	67.0	0	1	Yes	Private	U
1	51676	Female	61.0	0	0	Yes	Self-employed	I
2	31112	Male	80.0	0	1	Yes	Private	I
3	60182	Female	49.0	0	0	Yes	Private	U
4	1665	Female	79.0	1	0	Yes	Self-employed	I
...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	U
5106	44873	Female	81.0	0	0	Yes	Self-employed	U
5107	19723	Female	35.0	0	0	Yes	Self-employed	I
5108	37544	Male	51.0	0	0	Yes	Private	I
5109	44679	Female	44.0	0	0	Yes	Govt_job	U

5110 rows × 12 columns

In [4]:
```
1 df.isnull().sum().sum
```

Out[4]:
```
<bound method NDFrame._add_numeric_operations.<locals>.sum of id
0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64>
```

In [6]:
```
1 means = df.mean()
2 df.fillna(means, inplace = True)
3 df.to_csv(r"C:\Users\Anusha V\Downloads\archive (13)\healthcare-dataset
4 means
```

```
C:\Users\Anusha V\AppData\Local\Temp\ipykernel_22100\4153111646.py:1: Futu
reWarning: Dropping of nuisance columns in DataFrame reductions (with 'num
eric_only=None') is deprecated; in a future version this will raise TypeEr
ror.  Select only valid columns before calling the reduction.
  means = df.mean()
```

Out[6]:
```
id                  36517.829354
age                    43.226614
hypertension            0.097456
heart_disease           0.054012
avg_glucose_level     106.147677
bmi                    28.893237
stroke                  0.048728
dtype: float64
```

In [7]:
```
1 df.head()
```

Out[7]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_typ
0	9046	Male	67.0	0	1	Yes	Private	Urba
1	51676	Female	61.0	0	0	Yes	Self-employed	Rura
2	31112	Male	80.0	0	1	Yes	Private	Rura
3	60182	Female	49.0	0	0	Yes	Private	Urba
4	1665	Female	79.0	1	0	Yes	Self-employed	Rura

In [11]:
```
RM = df.drop_duplicates()
RM.to_csv(r"C:\Users\Anusha V\Downloads\archive (13)\healthcare-dataset
RM
```

Out[11]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_
0	9046	Male	67.0	0	1	Yes	Private	U
1	51676	Female	61.0	0	0	Yes	Self-employed	I
2	31112	Male	80.0	0	1	Yes	Private	I
3	60182	Female	49.0	0	0	Yes	Private	U
4	1665	Female	79.0	1	0	Yes	Self-employed	I
...	...	...	...	...	...	...	...	
5105	18234	Female	80.0	1	0	Yes	Private	U
5106	44873	Female	81.0	0	0	Yes	Self-employed	U
5107	19723	Female	35.0	0	0	Yes	Self-employed	I
5108	37544	Male	51.0	0	0	Yes	Private	I
5109	44679	Female	44.0	0	0	Yes	Govt_job	U

5110 rows × 12 columns

In [15]:
```
encoder = df['gender'] = df['gender'].replace({'Male': 0 , 'Female': 1}
encoder
```

Out[15]:
```
0       0
1       1
2       0
3       1
4       1
       ..
5105    1
5106    1
5107    1
5108    0
5109    1
Name: gender, Length: 5110, dtype: object
```

In [17]:
```
1 df.describe()
```

Out[17]:

	id	age	hypertension	heart_disease	avg_glucose_level	bm
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	5110.00000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.89323
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.69801
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.30000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.80000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.40000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	32.80000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.60000

In [19]:
```
1 groupby = df.groupby("age").count()
2 groupby
```

Out[19]:

age	id	gender	hypertension	heart_disease	ever_married	work_type	Residence_type	a
0.08	2	2	2	2	2	2	2	
0.16	3	3	3	3	3	3	3	
0.24	5	5	5	5	5	5	5	
0.32	5	5	5	5	5	5	5	
0.40	2	2	2	2	2	2	2	
...	...	...	...	...	...	...	...	
78.00	102	102	102	102	102	102	102	
79.00	85	85	85	85	85	85	85	
80.00	70	70	70	70	70	70	70	
81.00	60	60	60	60	60	60	60	
82.00	56	56	56	56	56	56	56	

104 rows × 11 columns

In [23]:
```
pivot = df.pivot(columns = 'age')
pivot
```

Out[23]:

	id														
age	**0.08**	**0.16**	**0.24**	**0.32**	**0.40**	**0.48**	**0.56**	**0.64**	**0.72**	**0.80**	**...**	**73.00**	**74.00**	**75.00**	**76.0**
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
5105	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5106	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5107	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5108	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5109	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

5110 rows × 1144 columns

In [22]:
```
df.dtypes
```

Out[22]:
```
id                   int64
gender              object
age                float64
hypertension         int64
heart_disease        int64
ever_married        object
work_type           object
Residence_type      object
avg_glucose_level  float64
bmi                float64
smoking_status      object
stroke               int64
dtype: object
```

In [24]:
```
df = df.drop(['work_type'], axis=1)
df
```

Out[24]:

	id	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_g
0	9046	0	67.0	0	1	Yes	Urban	
1	51676	1	61.0	0	0	Yes	Rural	
2	31112	0	80.0	0	1	Yes	Rural	
3	60182	1	49.0	0	0	Yes	Urban	
4	1665	1	79.0	1	0	Yes	Rural	
...	...	...	...	...	...	...	...	
5105	18234	1	80.0	1	0	Yes	Urban	
5106	44873	1	81.0	0	0	Yes	Urban	
5107	19723	1	35.0	0	0	Yes	Rural	
5108	37544	0	51.0	0	0	Yes	Rural	
5109	44679	1	44.0	0	0	Yes	Urban	

5110 rows × 11 columns

In [ ]:
```
1
```