WEI_KMeans

December 2, 2023

- import
- explore
- define groupby dataframes
  - k_cats, by Region, question category, 1971-2020
  - k_pivot, by question category, 2020
  - y_animate, by year for animation
- define functions
  - ElbowCurve(), plots a linechart with kmeans scores in range(k1, k2)
  - N_Clusters(), plots a barchart, adds 'k_cluster' column to df with n numbers of groups
  - N_Clusters_S(), plots a barchart, adds 'k_cluster' column to df with n numbers of groups from scaled data
  - PlotlyGroups(), plots 2D scatter plot and histogram, with trendline, colored grouping
  - PlotlyGroups3D(), plots 3D scatter plot, colored grouping
  - PlotlyGroups3D_Animate(), plots 3D scatter plot, colored grouping, animation variable
- KMeans Visualizations
  - initial exploration of k_pivot, 2D/3D scatterplots, and non-scaled kmeans clusters
- GDP, Life Exp, WEI Score
  - exploration of three main variables' kmeans clustering
- Life Exp, GDP, Population
- Year Animation (GDP, Index Score, Life Exp)
  - Animation through the years 1971-2020
- Conclusion
  - Thoughts and next steps

1 Import

- import libraries and data

[1]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import os
import plotly.express as px


path = r'/Users/amritambe/Desktop/Analysis_Project/Women_Empowerment'
```

```
df = pd.read_csv(os.path.join(path, '2 Data', 'Clean_Data',
 ↪'11_18_2023_WEI_CleanImputed.csv'), index_col='Unnamed: 0')
print('Data Imported:\n')
df.head()
```

```
Data Imported:
```

```
[1]:   Country_Name  Year  GDP_Growth  Index_1971  Index_2020  Fifty_Year_Change  \
0  Afghanistan  1971         0.0       210.0       305.0               95.0
1  Afghanistan  1971         0.0       210.0       305.0               95.0
2  Afghanistan  1971         0.0       210.0       305.0               95.0
3  Afghanistan  1971         0.0       210.0       305.0               95.0
4  Afghanistan  1971         0.0       210.0       305.0               95.0

       Region Income_Group Question_Category  \
0  South Asia   Low income            Assets
1  South Asia   Low income            Assets
2  South Asia   Low income            Assets
3  South Asia   Low income            Assets
4  South Asia   Low income            Assets

                                            Question  Index_Score  \
0  Do female and male surviving spouses have equa…          0.0
1  Do men and women have equal ownership rights t…         20.0
2  Do sons and daughters have equal rights to inh…          0.0
3  Does the law grant spouses equal administrativ…         20.0
4  Does the law provide for the valuation of nonm…          0.0

   2020_Data_Rank  2020_1GB_Price(USD)  Life_Exp  Population  GDP_Per_Cap  \
0            59.0                 1.55    36.088  13079460.0   739.981106
1            59.0                 1.55    36.088  13079460.0   739.981106
2            59.0                 1.55    36.088  13079460.0   739.981106
3            59.0                 1.55    36.088  13079460.0   739.981106
4            59.0                 1.55    36.088  13079460.0   739.981106

   Avg_WEI_Score
0       6.288571
1       6.288571
2       6.288571
3       6.288571
4       6.288571
```

2 Explore

- EDA
- df_k, df set to easily decide which variables to choose

- k, df set for the year 2020

[2]:
```
#create country id dataframe
country_id = pd.DataFrame(df['Country_Name'].unique()).reset_index()
country_id.columns = ['Country_Id', 'Country_Name']
country_id['Country_Id'] = country_id['Country_Id'] + 1
country_id
```

[2]:
```
     Country_Id         Country_Name
0             1          Afghanistan
1             2              Albania
2             3              Algeria
3             4               Angola
4             5  Antigua and Barbuda
..          …                    …
185         186              Vietnam
186         187            West Bank
187         188                Yemen
188         189               Zambia
189         190             Zimbabwe

[190 rows x 2 columns]
```

[3]:
```
#merge df to create country_id column
df_ = pd.merge(df, country_id, on='Country_Name', how='inner')
```

[4]:
```
df_.columns.tolist()
```

[4]:
```
['Country_Name',
 'Year',
 'GDP_Growth',
 'Index_1971',
 'Index_2020',
 'Fifty_Year_Change',
 'Region',
 'Income_Group',
 'Question_Category',
 'Question',
 'Index_Score',
 '2020_Data_Rank',
 '2020_1GB_Price(USD)',
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score',
 'Country_Id']
```

[5]:

```python
df_k = df_[[
 'Country_Id',
 'Country_Name',
 'Year',
 'GDP_Growth',
 'Index_1971',
 'Index_2020',
 'Fifty_Year_Change',
 'Region',
 'Income_Group',
 'Question_Category',
 'Question',
 'Index_Score',
 '2020_Data_Rank',
 '2020_1GB_Price(USD)',
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score'
       ]].copy()
```

3 Groupby Categories 1971-2020

- k_cats, df, categories for index scores by category and year

[6]:

```python
k_cats = df_k.groupby(['Question_Category', 'Year', 'Region'])['Index_Score'].
 ↪agg('mean').reset_index()
k_cats['Region_Code'] = k_cats['Region'].astype('category').cat.codes.
 ↪astype('int32')
k_cats.head()
```

[6]:

```
  Question_Category  Year                      Region  Index_Score  \
0            Assets  1971         East Asia & Pacific       12.960
1            Assets  1971       Europe & Central Asia       19.520
2            Assets  1971          High income: OECD       16.125
3            Assets  1971   Latin America & Caribbean       15.125
4            Assets  1971  Middle East & North Africa        8.600

   Region_Code
0            0
1            1
2            2
3            3
4            4
```

4 Groupby Question Category, 2020

- k_pivot, df of year 2020, grouping by question category

[7]:
```
k = df_k.query('Year == 2020').copy()
k.columns

kmeans = k.groupby(['Country_Name', 'Question_Category', 'Avg_WEI_Score',
                    'Life_Exp', 'Population', 'GDP_Per_Cap'])['Index_Score'] \
                    .agg('sum') \
                    .reset_index()

k_pivot = kmeans.pivot_table(index=['Country_Name', 'Life_Exp', 'Population',␣
 ↪'GDP_Per_Cap', 'Avg_WEI_Score'],
                             columns='Question_Category',
                             values='Index_Score') \
                             .reset_index()

k_pivot.sample(5)
```

[7]:
```
Question_Category     Country_Name  Life_Exp   Population    GDP_Per_Cap  \
187                          Yemen    62.698   22211743.0    2280.769906
107                         Mexico    76.195  108700891.0   11977.574960
41                         Denmark    78.332    5468120.0   35278.418740
150                Solomon Islands    77.926    2009245.0   25768.257590
100                         Malawi    48.303   13327079.0     759.349910

Question_Category  Avg_WEI_Score  Assets  Entrepreneurship  Marriage  \
187                     5.600000    40.0              75.0       0.0
107                    15.291429   100.0             100.0      60.0
41                     19.160000   100.0             100.0     100.0
150                    12.400000    80.0              75.0     100.0
100                    12.060000   100.0              75.0     100.0

Question_Category  Mobility  Parenthood    Pay  Pension  Workplace
187                    25.0         0.0   25.0     25.0       25.0
107                   100.0        60.0   75.0     75.0      100.0
41                    100.0       100.0  100.0    100.0      100.0
150                    75.0         0.0   25.0     75.0       25.0
100                    50.0        20.0  100.0    100.0      100.0
```

5 Groupby Year

- y_animate, df with all variables, Index Scores grouped by Year

[8]:
```
df_.columns.tolist()
```

[8]:

```
['Country_Name',
 'Year',
 'GDP_Growth',
 'Index_1971',
 'Index_2020',
 'Fifty_Year_Change',
 'Region',
 'Income_Group',
 'Question_Category',
 'Question',
 'Index_Score',
 '2020_Data_Rank',
 '2020_1GB_Price(USD)',
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score',
 'Country_Id']
```

[9]:

```
y_animate = df_.groupby(['Country_Name', 'Year',   'Region',
                         'Income_Group', 'Life_Exp', 'Population',
                         'GDP_Per_Cap'])['Index_Score'] \
                         .agg('sum') \
                         .reset_index()

# y_pivot = y_ani.pivot_table(index=['Country_Id', 'Year', '
#                                     ␣
 ↪'Index_1971','Index_2020','Fifty_Year_Change','Avg_WEI_Score',
#                                    'Life_Exp', 'Population', 'GDP_Per_Cap'],
#                             columns='Index_Score',
#                             values='Index_Score',
#                             aggfunc='sum') \
#                             .reset_index()

#change dtypes, make categorical codes for income and region
y_animate['Income_Category'] = y_animate['Income_Group'].astype('category') \
                                                        .cat.codes \
                                                        .astype('int32')
y_animate['Region_Category'] = y_animate['Region'].astype('category') \
                                                        .cat.codes \
                                                        .astype('int32')
print(y_animate.dtypes) #check dtypes

floats = ['Life_Exp','Population','GDP_Per_Cap','Index_Score']
y_animate[floats] = y_animate[floats].round().astype('int64')

print(y_animate.dtypes) #check dtypes
```

```
y_animate.sample(5)
```

```
Country_Name         object
Year                  int64
Region               object
Income_Group         object
Life_Exp            float64
Population          float64
GDP_Per_Cap         float64
Index_Score         float64
Income_Category       int32
Region_Category       int32
dtype: object
Country_Name        object
Year                 int64
Region              object
Income_Group        object
Life_Exp             int64
Population           int64
GDP_Per_Cap          int64
Index_Score          int64
Income_Category      int32
Region_Category      int32
dtype: object
```

[9]:

```
           Country_Name  Year                     Region         Income_Group  \
9247          Venezuela  2018  Latin America & Caribbean  Upper middle income
7044              Samoa  2015        East Asia & Pacific  Upper middle income
4033        Ivory Coast  2004         Sub-Saharan Africa  Lower middle income
6353   Papua New Guinea  1974        East Asia & Pacific  Lower middle income
5312          Mauritius  1983         Sub-Saharan Africa  Upper middle income

      Life_Exp  Population  GDP_Per_Cap  Index_Score  Income_Category  \
9247        76     3447496        10611          680                3
7044        46     8860588          863          640                3
4033        81    58147733        28570          555                2
6353        76     3242173         9809          385                2
5312        67      992040         3688          505                3

      Region_Category
9247                3
7044                0
4033                6
6353                0
5312                6
```

6 Groupby Country

- km_country, df made from y_animate to devise country specific k_clusters
- aim is to create country-specific k clusters that don't change through the years, for comparison

[10]:
```
km_country = y_animate.groupby(['Country_Name', 'Income_Category',␣
↪'Region_Category']) \
                              [['Life_Exp', 'GDP_Per_Cap', 'Index_Score']] \
                              .agg(['min', 'max', 'std', 'mean']) \
                              .reset_index()
km_country.columns.tolist()
```

[10]:
```
[('Country_Name', ''),
 ('Income_Category', ''),
 ('Region_Category', ''),
 ('Life_Exp', 'min'),
 ('Life_Exp', 'max'),
 ('Life_Exp', 'std'),
 ('Life_Exp', 'mean'),
 ('GDP_Per_Cap', 'min'),
 ('GDP_Per_Cap', 'max'),
 ('GDP_Per_Cap', 'std'),
 ('GDP_Per_Cap', 'mean'),
 ('Index_Score', 'min'),
 ('Index_Score', 'max'),
 ('Index_Score', 'std'),
 ('Index_Score', 'mean')]
```

[11]:
```
km_country.head()
```

[11]:
```
          Country_Name Income_Category Region_Category Life_Exp                \
                                                            min max       std
0          Afghanistan               1               5       36  44  2.634233
1              Albania               3               1       44  76  5.023861
2              Algeria               3               4       55  76  6.115354
3               Angola               2               6       38  72  4.695808
4  Antigua and Barbuda               0               3       43  43  0.000000

        GDP_Per_Cap                                Index_Score                \
    mean         min   max          std     mean         min  max        std
0  41.14         635   978   133.250761   824.50         205  305  26.678165
1  72.16         975  5937  1290.804780  4132.86         475  730  98.457543
2  66.48        4183  6223   668.988687  5423.88         260  460  73.215157
3  41.48        2277  6223  1224.904752  3602.32         245  585 125.130177
4  43.00        4797  4797     0.000000  4797.00         435  530  23.413758


    mean
```

```
0   220.1
1   570.7
2   344.5
3   409.7
4   512.4
```

7 ________________________________ defining viz functions ________________

8 ElbowCurve()

[12]:
```
k_pivot.columns.tolist()
```

[12]:
```
['Country_Name',
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score',
 'Assets',
 'Entrepreneurship',
 'Marriage',
 'Mobility',
 'Parenthood',
 'Pay',
 'Pension',
 'Workplace']
```

[13]:
```
    #k means for k_pivot df
    #implement Kmeans algo

    #Define function for K Means Elbow
def ElbowCurve(df, k1, k2):
        '''
        get k-means elbow curve using plotly
        - df, dataframe to use for kmeans
        - k1, low end of range for number of clusters
        - k2, high end of range for number of clusters, exclusive

        '''
        #decide on the variables:
        features_for_clustering = df

        scaler = StandardScaler()

        features_scaled = scaler.fit_transform(features_for_clustering)
```

```
        #Find optimal number of clusters
        k_values = range(k1, k2)  #any range here, right excluding
        kmeans = [KMeans(n_clusters=i, n_init=10) for i in k_values]

        #score the different cluster lengths
        score = [kmeans[i].fit(features_scaled).score(features_scaled) for i in␣
 ↪range(len(kmeans))]


        #plotly figure
        fig = px.line(x=k_values, y=score, markers=True,
                      title='Elbow Method for Optimal KMeans Clustering')

        fig.update_layout(xaxis_title='Number of Clusters (k)',
                          yaxis_title='Score',
                          xaxis=dict(tickmode='linear', dtick=1),
                          width=500, height=500)
        fig.show()
```

9 N_Clusters()

[14]:
```
#updates df
def N_Clusters(df, cols, n):
        '''
        - df, dataframe for kmeans predictions
        - cols, df columns with dtype(int, float)
        - n, number of clusters
        '''
        temp_df = df[cols].copy() #temporary df to work with only int/floats

        kmeans = KMeans(n_clusters=n, n_init=10)  #kmeans algo init
        temp_df['k_clusters'] = kmeans.fit_predict(temp_df)


        df = df.drop(columns=['k_clusters'], errors='ignore')    #delete␣
 ↪kcluster from old function calls

        df = pd.concat([df, temp_df['k_clusters']], axis=1)   #concatenate the␣
 ↪temp df with original, now with kcluster groupings


        #Use plotly to show value counts of new kmeans groups
        clusters = px.bar(df['k_clusters'].value_counts() \
                                          .sort_values(ascending=False),
                          title='K Means')
        clusters.update_layout(xaxis_title='Clusters (k)',
```

```
                  yaxis_title='Number',
                  width=500, height=500)
        clusters.show()
        return df
```

10 N_Clusters_S()

[15]:
```
#updates df
def N_Clusters_S(df, cols, n):
        '''
        Scaled data
        - df, dataframe for kmeans predictions
        - cols, df columns with dtype(int, float)
        - n, number of clusters
        '''
        temp_df = df[cols].copy() #temporary df to work with only int/floats

        #scale data
        scaler = StandardScaler()

        temp_df_scaled = scaler.fit_transform(temp_df)

        #fit kmeans
        kmeans = KMeans(n_clusters=n, n_init=10)  #kmeans algo init

        temp_df['k_clusters'] = kmeans.fit_predict(temp_df_scaled)


        #Merge k_cluster to df
        df = df.drop(columns=['k_clusters'], errors='ignore')    #delete␣
 ↪kcluster from old function calls

        df = pd.concat([df, temp_df['k_clusters']], axis=1)   #concatenate the␣
 ↪temp df with original, now with kcluster groupings


        #Use plotly to show value counts of new kmeans groups
        clusters = px.bar(df['k_clusters'].value_counts() \
                                          .sort_values(ascending=False),
                          title='K Means Scaled')
        clusters.update_layout(xaxis_title='Clusters (k)',
                  yaxis_title='Number',
                  width=500, height=500)
        clusters.show()
        return df
```

11 PlotlyGroups()

[16]:
```
# PlotlyGroups(k_pivot, 'Life_Exp', 'Assets', 'k_clusters')

def PlotlyGroups(df, x, y, color=None):
        '''
        - df, dataframe
        - x, xaxis
        - y, yaxis
        - color, Groups to color by, columns
        '''
        fig2 = px.scatter(df, x, y,
                          color=color, color_continuous_scale=['red', 'green',␣
 ↪'blue', 'gray'],
                          trendline='ols', marginal_x='histogram',
                          title='K Means Clusters')
        # Update marker color for all traces and layout
        fig2.update_layout(width=700, height=700,
                          showlegend=False)
        fig2.show()
```

12 PlotlyGroups3D()

[17]:
```
def PlotlyGroups3D(df, x, y, z, color, hover):
        '''
        - df, dataframe
        - x, xaxis
        - y, yaxis
        - z, zaxis
        - color, groupings (df column)
        - hover, info data (df columns)
        '''
        fig3 = px.scatter_3d(df, x, y, z,
                          color=color,
                          hover_name=hover,
                          color_continuous_scale=['red', 'green', 'blue',␣
 ↪'gray'],
                          title='K Means Clusters')
        # Update marker color for all traces and layout
        fig3.update_layout(scene=dict(
                            xaxis_title=x,
                            yaxis_title=y,
                            zaxis_title=z),
                            width=1000, height=700,
                            showlegend=False)
        fig3.show()
```

13 PlotlyGroups3D__Animate()

[18]:
```
def PlotlyGroups3D_Animate(df, x, y, z, color, size=None, hover=None,␣
↪animate=None):
        '''
        - df, dataframe
        - x, xaxis
        - y, yaxis
        - z, zaxis
        - color, group variable (df column)
        - size, size variable (df column)
        - hover, info variable (df column) [can be list]
        - animate, animated variable (df column)
        '''
        fig3 = px.scatter_3d(df, x, y, z,
                             color=color,
                             hover_data=hover,
                             animation_frame=animate,
                             color_continuous_scale=['green', 'red', 'blue',␣
↪'orange'],
                             title='K Means Clusters')
        # Update marker color for all traces and layout
        fig3.update_layout(scene=dict(
                               xaxis=dict(range=[0, df[x].max()]),
                               yaxis=dict(range=[0, df[y].max()]),
                               zaxis=dict(range=[0, df[z].max()]),
                               xaxis_title=x,
                               yaxis_title=y,
                               zaxis_title=z),
                               width=1000, height=700,
                               showlegend=False)

        fig3.show()
```

14 ____ Visualizations ________________________________

15 KMeans Visualizations

[19]:
```
k_pivot.columns.tolist()
```

[19]:
```
['Country_Name',
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score',
```

```
 'Assets',
 'Entrepreneurship',
 'Marriage',
 'Mobility',
 'Parenthood',
 'Pay',
 'Pension',
 'Workplace']
```

[20]:
```
kcol = [
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score',
 'Assets',
 'Entrepreneurship',
 'Marriage',
 'Mobility',
 'Parenthood',
 'Pay',
 'Pension',
 'Workplace']
```

[21]:
```
ElbowCurve(k_pivot[kcol], 1, 16)
```

- It seems that 4 or 5 clusters are optimal, lets check and update the dataframe:

[22]:
```
k_un = N_Clusters(k_pivot, kcol, 4) #without scaling data
k_sc = N_Clusters_S(k_pivot, kcol, 4) #with scaling data
```

[23]:
```
k_sc.columns
```

[23]:
```
Index(['Country_Name', 'Life_Exp', 'Population', 'GDP_Per_Cap',
       'Avg_WEI_Score', 'Assets', 'Entrepreneurship', 'Marriage', 'Mobility',
       'Parenthood', 'Pay', 'Pension', 'Workplace', 'k_clusters'],
      dtype='object')
```

[24]:
```
PlotlyGroups(k_sc, 'Life_Exp', 'Avg_WEI_Score', 'k_clusters')
```

[25]:
```
PlotlyGroups3D(k_sc, 'Life_Exp', 'Avg_WEI_Score', 'GDP_Per_Cap', 'k_clusters',␣
 ↪'Country_Name')
```

16 GDP, Life Exp, WEI Score

[26]:
```
k_pivot.columns.tolist()
```

[26]: ['Country_Name',
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score',
 'Assets',
 'Entrepreneurship',
 'Marriage',
 'Mobility',
 'Parenthood',
 'Pay',
 'Pension',
 'Workplace']

[27]:
```
k_p = [
    'GDP_Per_Cap',
    'Avg_WEI_Score',
    'Life_Exp'
]
```

[28]:
```
ElbowCurve(k_pivot[k_p], 1, 11)
```

[29]:
```
k_  = N_Clusters(k_pivot, k_p, 4)
k_scaled = N_Clusters_S(k_pivot, k_p, 4)
```

Here, non scaled data results in a kmeans clustering that reflects GDP per Cap strongly, while scaled data

[30]:
```
print('First the unscaled clusters:')
PlotlyGroups3D(k_, 'GDP_Per_Cap', 'Avg_WEI_Score', 'Life_Exp', 'k_clusters',␣
 ↪'Country_Name')
print('\nNow the clusters using scaled data:')
PlotlyGroups3D(k_scaled, 'GDP_Per_Cap', 'Avg_WEI_Score', 'Life_Exp',␣
 ↪'k_clusters', 'Country_Name')
```

```
First the unscaled clusters:


Now the clusters using scaled data:
```

It appears that the unscaled data is mostly marking 4 groups by gdp per capita, <6k, 6k-15k, 15k-30k, >30k

The scaled data is showing a similar affinity for grouping by gdp, with a 4th group having overall having *low empowerment scores* relative to their peers in gdp per capita.

17 Life Exp, GDP, Pop

[31]:
```
k_pivot.columns.tolist()
```

[31]:
```
['Country_Name',
 'Life_Exp',
 'Population',
 'GDP_Per_Cap',
 'Avg_WEI_Score',
 'Assets',
 'Entrepreneurship',
 'Marriage',
 'Mobility',
 'Parenthood',
 'Pay',
 'Pension',
 'Workplace']
```

[32]:
```
#choose k_pivot quantitative variables
lgp = [
 'Life_Exp',
 'Population',
 'GDP_Per_Cap'
]
```

[33]:
```
ElbowCurve(k_pivot[lgp], 1, 11)
```

[34]:
```
lgp_ = N_Clusters(k_pivot, lgp, 4)
lgp_scaled = N_Clusters_S(k_pivot, lgp, 4)
```

[35]:
```
lgp_.columns
```

[35]:
```
Index(['Country_Name', 'Life_Exp', 'Population', 'GDP_Per_Cap',
       'Avg_WEI_Score', 'Assets', 'Entrepreneurship', 'Marriage', 'Mobility',
       'Parenthood', 'Pay', 'Pension', 'Workplace', 'k_clusters'],
      dtype='object')
```

[36]:
```
PlotlyGroups3D(lgp_,  'Life_Exp', 'Population', 'GDP_Per_Cap', 'k_clusters',␣
 ↪'Country_Name')
PlotlyGroups3D(lgp_scaled,  'Life_Exp', 'Population', 'GDP_Per_Cap',␣
 ↪'k_clusters', 'Country_Name')
```

[ ]:

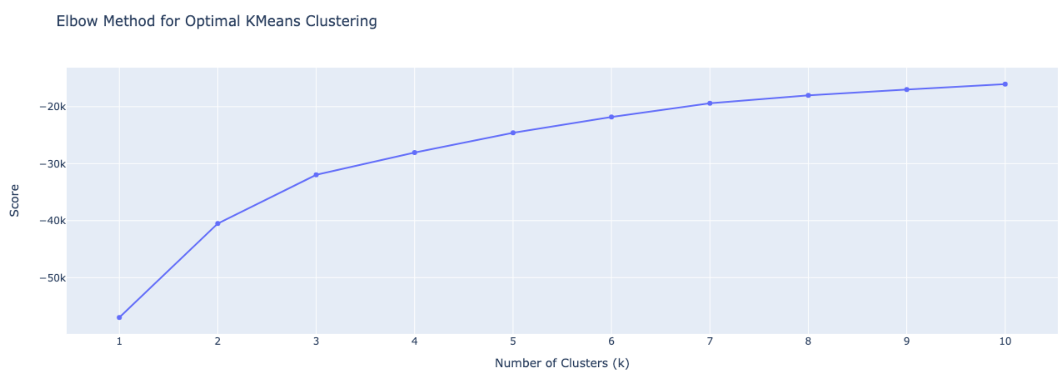
18 Year Animation

[37]:
```
y_animate.columns
```

[37]:
```
Index(['Country_Name', 'Year', 'Region', 'Income_Group', 'Life_Exp',
       'Population', 'GDP_Per_Cap', 'Index_Score', 'Income_Category',
       'Region_Category'],
      dtype='object')
```

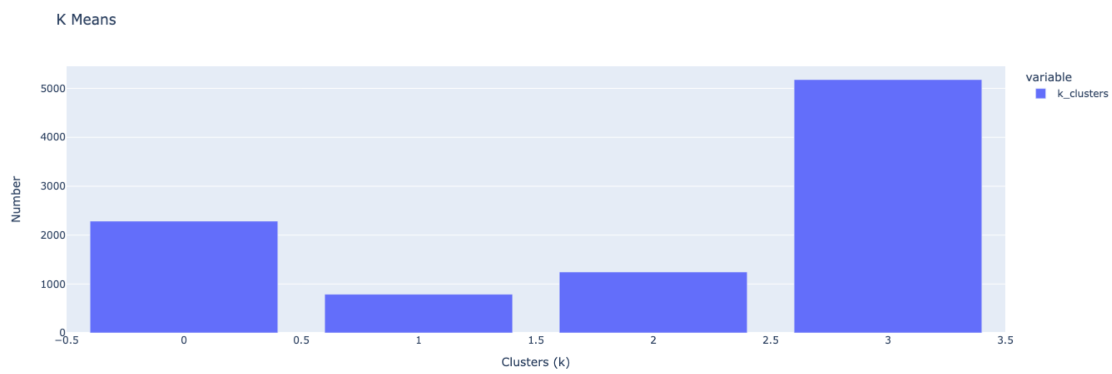
[38]:
```
#variables to use for kmeans. Must be int/float
y_var = ['GDP_Per_Cap', 'Index_Score', 'Life_Exp', 'Year', 'Income_Category',␣
 ↪'Region_Category']
```

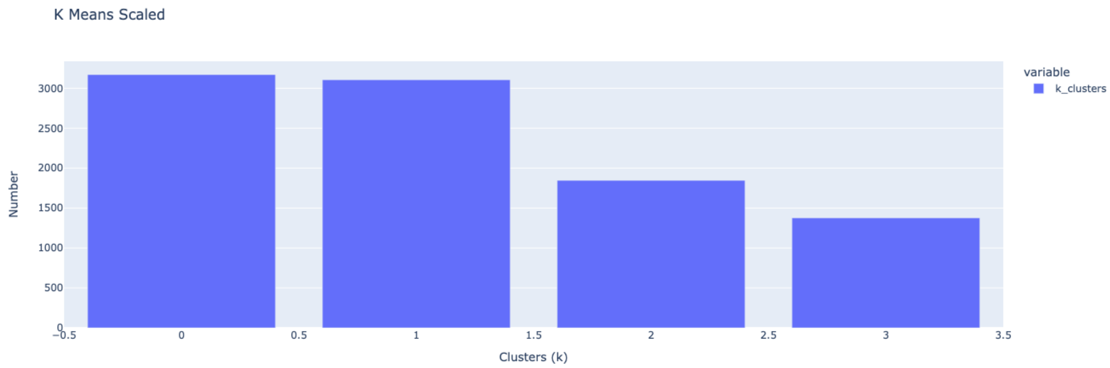
[39]:
```
ElbowCurve(y_animate[y_var], 1, 11)
```



It looks like the optimal number of clusters is 4.

[40]:
```
y_ = N_Clusters(y_animate, y_var, 4)    #kmeans using unprocessed data
y_scaled = N_Clusters_S(y_animate, y_var, 4)    #kmeans using scaled data
```

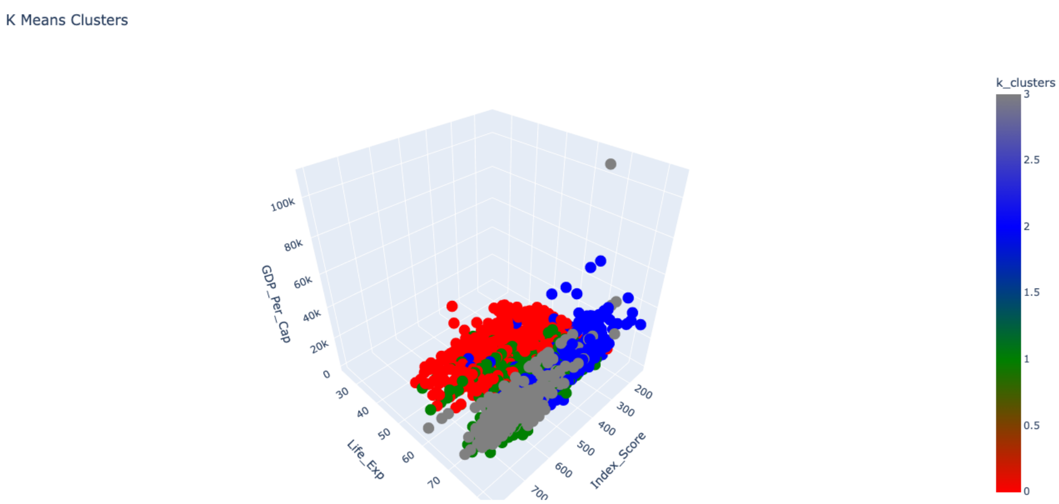




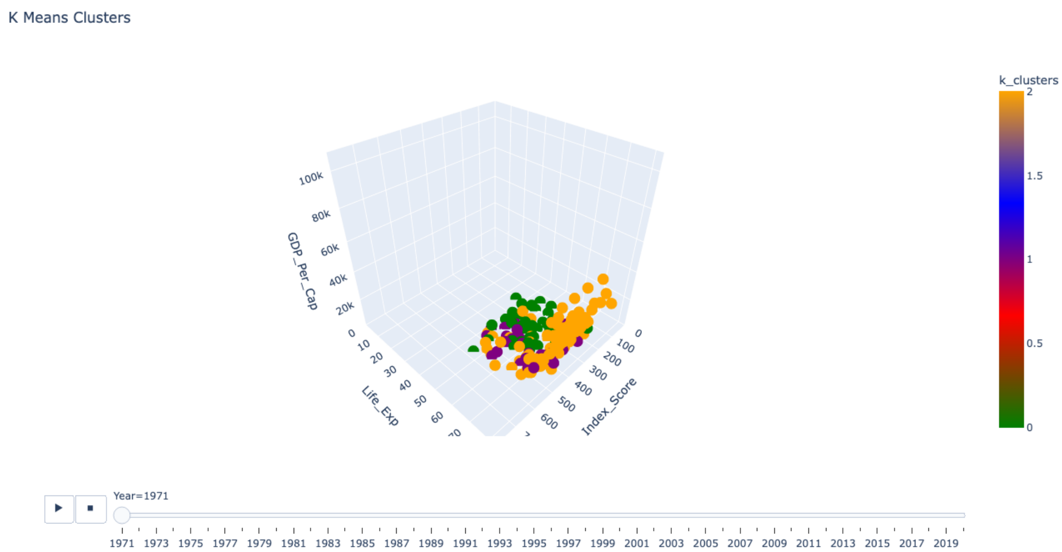
Using both scaled and unscaled kmeans groups can be useful.

- y__ is unscaled, y__scaled is scaled.

[41]:
```
PlotlyGroups3D(y_scaled, 'Index_Score', 'Life_Exp',  'GDP_Per_Cap',␣
↪'k_clusters', 'Country_Name')
```



[42]:
```
PlotlyGroups3D_Animate(y_scaled, 'Index_Score', 'Life_Exp',  'GDP_Per_Cap',␣
↪'k_clusters',
                       hover=['Country_Name', 'Region', 'Income_Group'],␣
↪animate='Year')
```



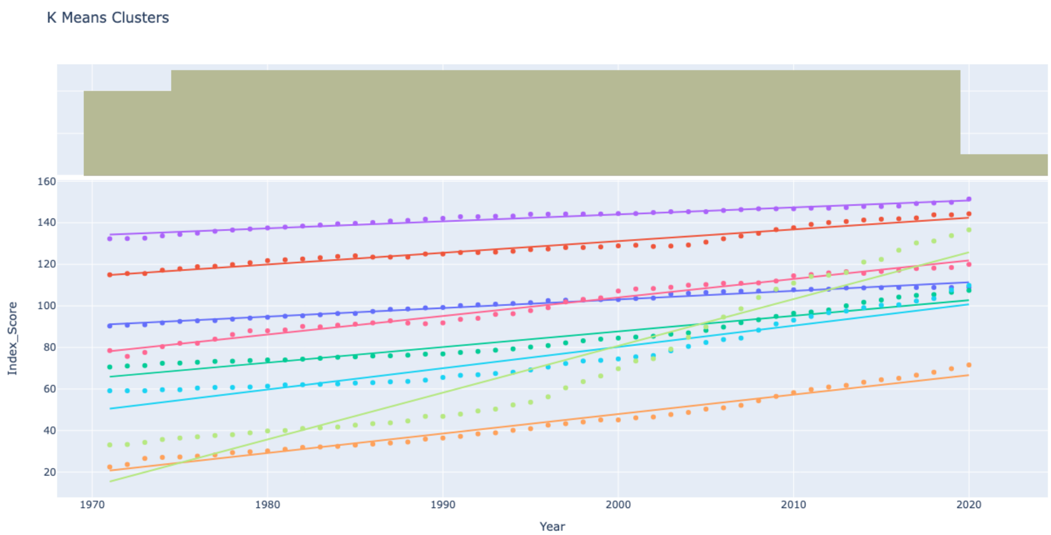
Lets see the animation of the index categories by year, using k_cats

[43]:
```
k_cat_group = k_cats.groupby(['Question_Category', 'Year'])['Index_Score'].
    ↪agg('sum').reset_index()
k_cat_group.head()
```

[43]:

	Question_Category	Year	Index_Score
0	Assets	1971	90.330000
1	Assets	1972	90.746667
2	Assets	1973	90.913333
3	Assets	1974	91.830000
4	Assets	1975	92.525000

[44]:
```
PlotlyGroups(k_cat_group, 'Year', 'Index_Score', 'Question_Category')
```



19 Country K Clusters

The goal here is to create k clusters that are grouped on individual countries, set on y_animate_c, and compare the animation through the years with k clusters that are not country specific.

[45]:
```
cntry = [
 ('Income_Category', ''),
 ('Region_Category', ''),
 ('Life_Exp', 'min'),
 ('Life_Exp', 'max'),
 ('Life_Exp', 'std'),
 ('Life_Exp', 'mean'),
 ('GDP_Per_Cap', 'min'),
 ('GDP_Per_Cap', 'max'),
 ('GDP_Per_Cap', 'std'),
 ('GDP_Per_Cap', 'mean'),
 ('Index_Score', 'min'),
 ('Index_Score', 'max'),
 ('Index_Score', 'std'),
 ('Index_Score', 'mean')
]


ElbowCurve(km_country[cntry], 1, 11)     #4 clusters seem appropriate

km_ = N_Clusters(km_country, cntry, 4)       #Scaled and Unscaled clusters
```

```
km_scaled = N_Clusters_S(km_country, cntry, 4)      # Scaled clusters are better
```

[46]:
```
km_scaled['Country_Name'] = km_scaled[('Country_Name', '')]  #rename, prepping␣
↪for merge

#now we merge km_scaled['k_clusters'] and y_animate on 'Country_Name'
print('shape of two df before merge:', y_animate.shape, km_scaled.columns,␣
↪km_scaled.shape)  #checks, merge,
y_animate_c = pd.merge(y_animate, km_scaled[['Country_Name', 'k_clusters']],
                       how='inner', on='Country_Name',
                       indicator=True)
# print('shape of merged df:', y_animate_c.shape)   #check merge col, drop col
print('\nmerge counts:\n', y_animate_c['_merge'].value_counts(), y_animate_c.
↪shape)
y_animate_c.drop(columns='_merge', axis=1, inplace=True)
y_animate_c = y_animate_c[y_animate_c['Year'] != 1971]   #delete erroneous data␣
↪from 1971 (gapminder data starts in 72)
print('merge col, year 1971 dropped', y_animate_c.columns) #checks
```

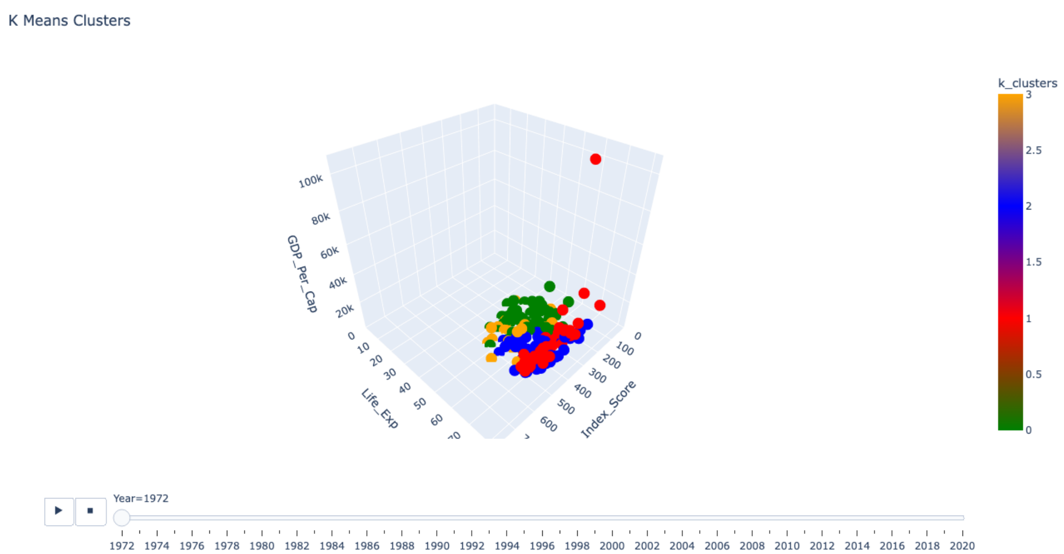
```
shape of two df before merge: (9500, 10) Index([   ('Country_Name', ''),
('Income_Category', ''),
       ('Region_Category', ''),     ('Life_Exp', 'min'),
           ('Life_Exp', 'max'),     ('Life_Exp', 'std'),
          ('Life_Exp', 'mean'),  ('GDP_Per_Cap', 'min'),
        ('GDP_Per_Cap', 'max'),  ('GDP_Per_Cap', 'std'),
       ('GDP_Per_Cap', 'mean'),  ('Index_Score', 'min'),
        ('Index_Score', 'max'),  ('Index_Score', 'std'),
       ('Index_Score', 'mean'),            'k_clusters',
                'Country_Name'],
      dtype='object') (190, 17)

merge counts:
 both          9500
left_only        0
right_only       0
Name: _merge, dtype: int64 (9500, 12)
merge col, year 1971 dropped Index(['Country_Name', 'Year', 'Region',
'Income_Group', 'Life_Exp',
       'Population', 'GDP_Per_Cap', 'Index_Score', 'Income_Category',
       'Region_Category', 'k_clusters'],
      dtype='object')
```

[47]:
```
PlotlyGroups3D_Animate(y_animate_c, 'GDP_Per_Cap', 'Life_Exp', 'Index_Score',
                       'k_clusters', hover=['Country_Name'], animate='Year')
```

20 Conclusions

21 4 Clusters over the Years

[48]:
```
PlotlyGroups3D_Animate(y_animate_c, 'Index_Score', 'Life_Exp',  'GDP_Per_Cap', color='k_clusters',
                           hover=['Country_Name', 'Region', 'Income_Group'], animate='Year')
```



22 4 main Groups:

groups (1, 3) and (0, 2) are worth analyzing further

(1, 3): in 1970s groups 1 is head and shoulders above the rest re: life expectancy and gdp per capita, but have one differentiating factor-Index Score– with group 3.

from 2000s on group 3 takes the lead, but a huge factor is growth. group 1 basically remained static re: life exp and gdp. further study of these groups and underlying differences and outcomes will be needed.

(0, 2): Both groups struggle with low gdp per capita, where they differ is life exp/ index score.

group 0 - they are ahead in gdp per capita vs group 2, but those countries that overperform tend to have high index scores group 1 - struggle on all variables. even with relatively high index scores, low gdp and life exp.

23 descriptive stats for k clusters

- The clusters make sense when visualizing the countries' trajectory through time, however some of the stats have quite a lot of overlap.
- Group 0 and 2 have similar profiles.
  - Group 2 has remarkaby low mean life expectancy (54)
  - Group 0 has gdp and life profile similar to Group 2, but much better index scores (highest min index score, 2nd higheset mean)
- Group 1 and 3 have similar profiles.
  - Group 1 has highest GDP per capita by a wide margin
  - Group 3 has the highest avg index score and highest max gdp per capita

[49]:
```
#aggregate statistics for the k clusters:
k_stats = y_animate_c.groupby(['k_clusters'])[['Life_Exp', 'GDP_Per_Cap',␣
↪'Index_Score']] \
                        .agg(['min', 'max', 'std', 'mean', 'median']) \
                        .T
k_stats
```

[49]:

k_clusters		0	1	2	3
Life_Exp	min	24.000000	63.000000	50.000000	36.000000
	max	79.000000	83.000000	79.000000	64.000000
	std	9.781977	3.527054	5.148599	5.860364
	mean	55.567055	76.715154	70.839703	52.442602
	median	55.000000	77.000000	72.000000	53.000000
GDP_Per_Cap	min	241.000000	3031.000000	677.000000	347.000000
	max	34168.000000	109348.000000	25768.000000	13206.000000
	std	5585.881772	11894.666917	5084.568296	3015.468999
	mean	3976.705904	25821.142423	7442.900186	2014.896046
	median	1483.000000	25116.000000	6466.000000	1043.000000
Index_Score	min	140.000000	185.000000	210.000000	140.000000
	max	705.000000	800.000000	775.000000	780.000000
	std	127.706601	136.612093	114.023240	115.629488
	mean	379.323980	578.043856	492.269017	461.192602
	median	370.000000	585.000000	485.000000	460.000000

Next steps and analysis:

1) Uncover key disparities between Groups 1 and 3 in the 1970s, focusing on Group 3's surge post-2000.

2) Analysis of Groups 0 and 2 regarding Index Scores and GDP.

3) Scrutinize outliers within clusters for potential success stories or areas requiring attention.

4) Integrate insights into a forward-looking scenario planning exercise to optimize decision strategies.

24 export visualizations

[50]:
```
#groupby region for gdp per cap
y_region = y_scaled.query('Year < 2007 and Year > 1972') \
                   .groupby(['Year', 'Region'])['GDP_Per_Cap'] \
                   .agg('mean') \
                   .reset_index()
#groupby region for life exp
y_lifeexp = y_scaled.query('Year < 2007 and Year > 1972') \
                   .groupby(['Year', 'Region'])['Life_Exp'] \
                   .agg('mean') \
                   .reset_index()
#groupby region for index score
y_score = y_scaled.query('Year < 2007 and Year > 1972') \
                   .groupby(['Year', 'Region'])['Index_Score'] \
                   .agg('mean') \
                   .reset_index()



#chart GDP per cap
gdp_r = px.line(y_region, 'Year', 'GDP_Per_Cap',
               line_group='Region',
               color='Region',
               title='GDP Per Capita by Region: 1972-2007',
               width=800, height=500)
gdp_r.show()

#chart life exp
life_r = px.line(y_lifeexp, 'Year', 'Life_Exp',
               line_group='Region',
               color='Region',
               title='Life Expectancy by Region: 1972-2007',
               width=800, height=500)
life_r.show()

#chart index score
score_r = px.line(y_score, 'Year', 'Index_Score',
               line_group='Region',
               color='Region',
               title="Women's Empowerment Score by Region: 1972-2007",
               width=800, height=500)
score_r.show()
```

[51]:
```
# life_r.write_image(os.path.join(path, '4 Analysis', 'Regional_Life_Exp.png'))

# gdp_r.write_image(os.path.join(path, '4 Analysis', 'Regional_GDP_percap.png'))
```

```
# score_r.write_image(os.path.join(path, '4 Analysis', 'Regional_WEI_Score.
↪png'))
```

25 export Data

[52]:
```
y_animate_c.shape, y_animate_c.columns
```

[52]:
```
((9310, 11),
 Index(['Country_Name', 'Year', 'Region', 'Income_Group', 'Life_Exp',
        'Population', 'GDP_Per_Cap', 'Index_Score', 'Income_Category',
        'Region_Category', 'k_clusters'],
       dtype='object'))
```

[53]:
```
df_k.shape, df_k.columns
```

[53]:
```
((332500, 18),
 Index(['Country_Id', 'Country_Name', 'Year', 'GDP_Growth', 'Index_1971',
        'Index_2020', 'Fifty_Year_Change', 'Region', 'Income_Group',
        'Question_Category', 'Question', 'Index_Score', '2020_Data_Rank',
        '2020_1GB_Price(USD)', 'Life_Exp', 'Population', 'GDP_Per_Cap',
        'Avg_WEI_Score'],
       dtype='object'))
```

[54]:
```
y_c = y_animate_c[['Country_Name', 'k_clusters']].drop_duplicates()   #prep␣
↪data into just country_name, k cluster

k_cluster_df = pd.merge(df_k, y_c[['Country_Name', 'k_clusters']],        #merge␣
↪dfs
                        how='left', on='Country_Name',
                        indicator=True)

print('Merge value counts:\n', k_cluster_df['_merge'].value_counts(),␣
↪'\n\nshape:', k_cluster_df.shape)     #print checks

k_cluster_df.drop(columns='_merge', axis=1, inplace=True)       #drop _merge␣
↪column
print('\nMerge col dropped:', k_cluster_df.shape)

k_cluster_df = k_cluster_df[k_cluster_df['Year'] != 1971]    #delete data from␣
↪1971, erroneous data
print('\nnew shape after dropping erroneous 1971 data:', k_cluster_df.shape)
```

```
Merge value counts:
 both          332500
left_only          0
right_only         0
Name: _merge, dtype: int64
```

```
shape: (332500, 20)

Merge col dropped: (332500, 19)

new shape after dropping erroneous 1971 data: (325850, 19)
```

[55]:
```
#export new df-- k clusters included and no erroneous 1971 data

k_cluster_df.to_csv(os.path.join(path, '2 Data', 'Clean_Data',␣
 ↪'12_01_2023_Kmeans_WEI_.csv'), encoding='utf-8')
```

Check errors in exporting. - it seems encoding needs to be set to utf-8

[56]:
```
what = pd.read_csv(os.path.join(path, '2 Data', 'Clean_Data',␣
 ↪'12_01_2023_Kmeans_WEI_.csv'))

india_b = k_cluster_df.query('Country_Name == "India"')['k_clusters'].mean()
india_a = what.query('Country_Name == "India"')['k_clusters'].agg('mean')


#print checks
print('\nIndia k cluster before exporting:', india_b )
print('India k cluster after exporting:', india_a)

china_b = k_cluster_df.query('Country_Name == "China"')['k_clusters'].mean()
china_a = what.query('Country_Name == "China"')['k_clusters'].agg('mean')


#print checks
print('\nChina k cluster before exporting:', china_b )
print('China k cluster after exporting:', china_a)
```

```
India k cluster before exporting: 0.0
India k cluster after exporting: 0.0

China k cluster before exporting: 2.0
China k cluster after exporting: 2.0
```

[ ]:
```

```