Data Analysis with Python

- *Tharindu Prabhath*

Projects >>

1 - Mean Variance

2 - Demographic Data Analyzer

3 - Medical Data Visualizer

4 - Time Series Visualizer

5 - Sea Level Predictor

In [70]:
```
import numpy as np
import pandas as pd

def calculate(n):

    if len(n) == 9:
        r = n.reshape(3,3)  # r is the reshaped matrix of the user input matrix (n)

        Sum=[]
        Min=[]
        Max=[]
        Mean=[]
        SD=[]
        Var=[]

        for i in range(3):

            Sum.append(float(r[[i],:].sum()))
            Min.append(float(r[[i],:].min()))
            Max.append(float(r[[i],:].max()))
            Mean.append(float(r[[i],:].mean()))
            SD.append(float(r[[i],:].std()))
            Var.append(float(r[[i],:].var()))


        rows=[Sum,Min,Max,Mean,SD,Var]
        df=pd.DataFrame(rows,columns=['Row 1','Row 2','Row 3'],index=['Summation','Minumum','Maximum','Mean','STD','Variance']
        return df



    else:
        print('The array should be a row matrix with 9 elements.... So please re enter ! .....')
```

In [71]:
```
n=np.array([1,2,3,4,5,6,7,8,9])
calculate(n)
```

Out[71]:

	Row 1	Row 2	Row 3
Summation	6.000000	15.000000	24.000000
Minumum	1.000000	4.000000	7.000000
Maximum	3.000000	6.000000	9.000000
Mean	2.000000	5.000000	8.000000
STD	0.816497	0.816497	0.816497
Variance	0.666667	0.666667	0.666667

In [ ]:

In [ ]:

> Importing packages and loading the dataset >>

In [27]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df=pd.read_csv('adult.data.csv')
# loading a dataset ( in csv format )

df.head()
# showing 1st 5 row of the dataset
```

Out[27]:

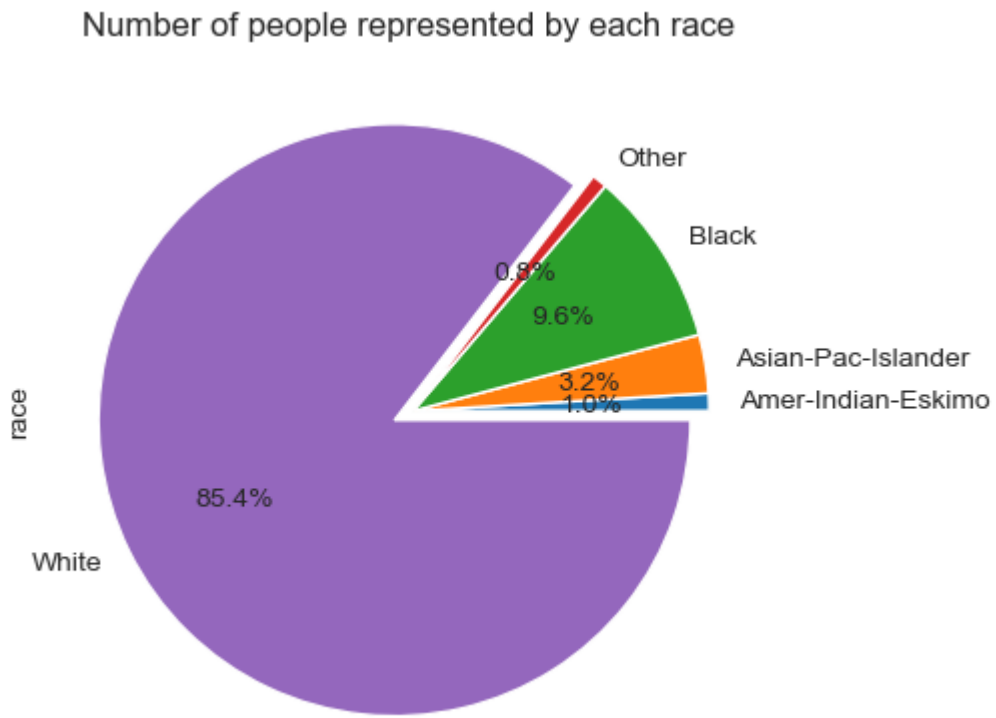
	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba

> Number pf people representing by each race >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In [28]:
```
# i am declared a color matrix : for later use
col=['orangered','navy','chartreuse','darkviolet','tomato','tan','purple','coral','green','deeppink','magenta','red','skyblue'

dfg1=df.groupby('race')['race'].count()
# data grouping according to the each race and then counting
dfg1
```

Out[28]:
```
race
Amer-Indian-Eskimo      311
Asian-Pac-Islander     1039
Black                  3124
Other                   271
White                 27816
Name: race, dtype: int64
```

In [29]:
```
dfg1.plot.pie(
    color=col, # calling the previously defined color array
    explode=(0,0,0,0,0.07), # exploding the selected slices
    autopct='%1.1f%%'  # percentage view
              )
plt.title('Number of people represented by each race ',fontsize=12,pad=2)
# font size and padding changing
plt.show()
```



> Avarage age of men >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In [30]:
```
print( 'Avarage age of men : ',int(  df[df['sex']=='Male']['age'].mean() ) ,' ( as a integer ) ' )
# filtering Males > then taking the avarage of the age column ( filtered data )
```

```
Avarage age of men :  39  ( as a integer )
```

> Percentage of people who have Bachelor's degree >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In [31]:
```
bachelors_count = len( df[df['education'] == 'Bachelors'].index )
# len() returns the lenght of the array
total_count = len(df['education'])

print( "\n  Percentage of people who have Bachelor's degree : ",
      round( bachelors_count/total_count * 100 , 2) ,' %   \n'  )
# in here round() is used for round the value upto  02 decimal points
```

```
 Percentage of people who have Bachelor's degree :  16.45  %
```

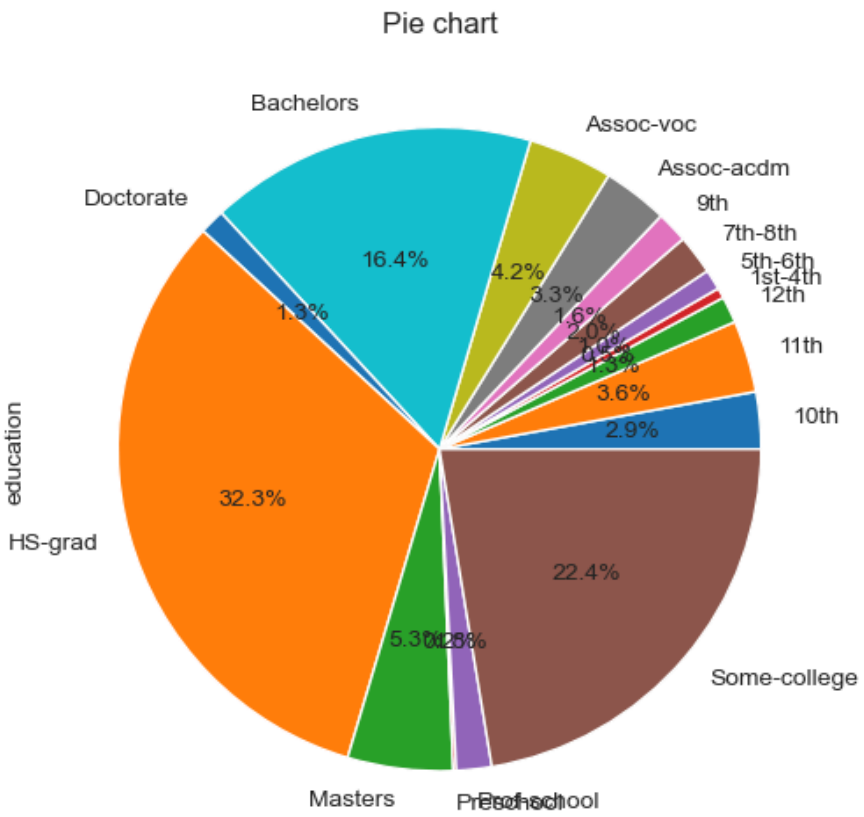
> Percentage of people with advanced education (Bachelor's, Masters or Doctorate)
> >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

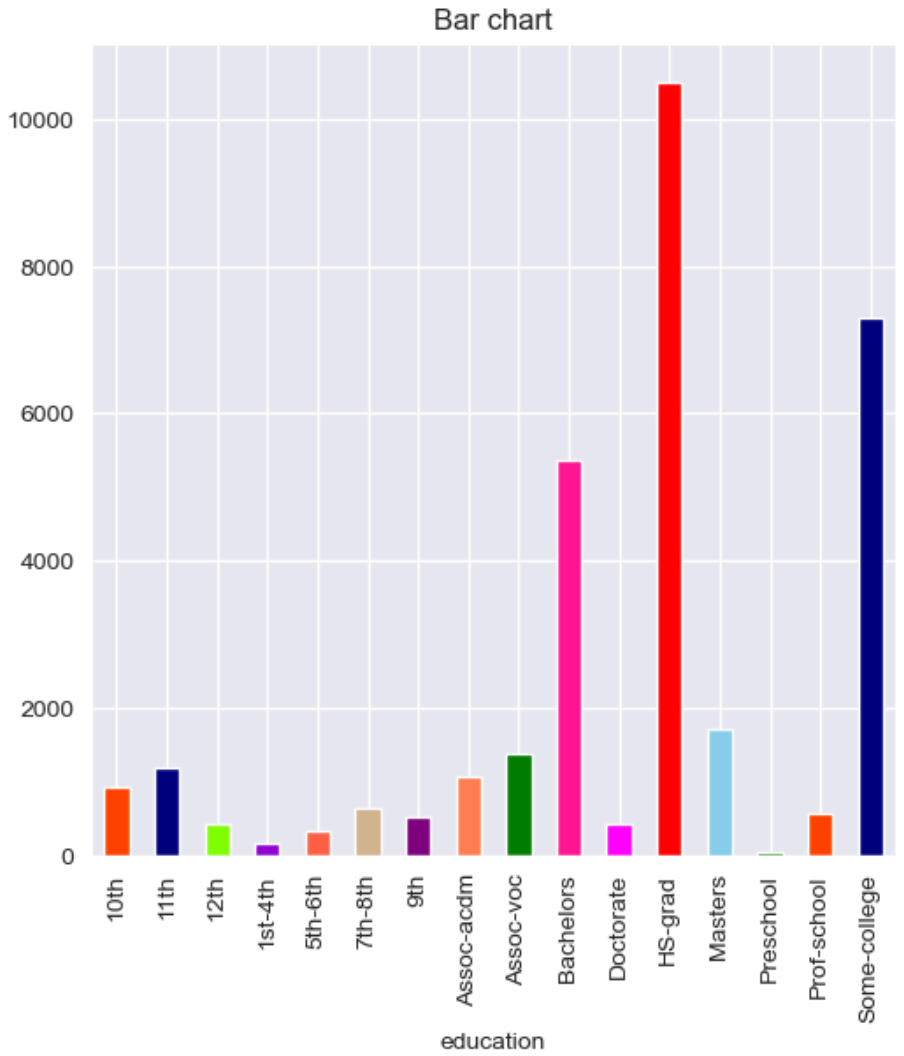
In [32]:
```
# isin() can be used to filter data
adv_edu_count = len(
    df[df['education'].isin( [ 'Bachelors','Masters','Doctorate' ] ) ]['education']
)

print('\n  Percentage of people with advanced education : ',
      round( adv_edu_count / total_count * 100 , 2 ) , ' %  \n' )
```

```
 Percentage of people with advanced education :  23.01  %
```

In [33]:
```
plt.figure(figsize=(13,6)) # to create a figure of the size 13 X 6
sns.set_style('darkgrid')
# this 'seaborn' command can be used to change the background style of the plots

plt.subplot(121)
# this plot will be displayed as the 1st chart in the figure of  1 x 2
df.groupby('education')['education'].count().plot.pie(
        color=col,
        autopct='%1.1f%%'
)
plt.title('Pie chart')

plt.subplot(122)
```

```
# this plot will be displayed as the 2nd chart in the figure of  1 x 2
df.groupby('education')['education'].count().plot.bar(
    color=col
)

plt.title('Bar chart')

plt.show()
```





> Percentage of people without advanced education make more than 50K >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In [34]:
```
# ' ~ ' is using for indicate the 'NOT' logic and ' & ' is using for 'AND' logic
no_adv_50k_count = len(
    df[
    ~ df['education'].isin( [ 'Bachelors','Masters','Doctorate' ] ) &
    (df['salary'] == '>50K')  ].index
)

print('\n Percentage of people without advanced education make more than 50K : ',
     round( no_adv_50k_count / total_count *100 , 2 ) , ' %  \n')
```

```
 Percentage of people without advanced education make more than 50K :  13.37  %
```

> Minimum number of hours a person works per week >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In [35]:
```
print('\n Minimum number of hours a person works per week : ',
      df['hours-per-week'].min() ,'\n')
```

```
 Minimum number of hours a person works per week :  1
```

> Percentage of the people who work min number of hours per week have a salary more than 50K >>>>>>>>>>>>>>>>>>>>

In [36]:
```
min_count = len( df[df['salary'].isin(['>50K']) & (df['hours-per-week']==1)].index )

print('\n Percentage of the people who work min number of hours per week have a salary more than 50K : ',
      round( min_count/total_count * 100 , 2) , ' %  \n')
```

```
 Percentage of the people who work min number of hours per week have a salary more than 50K :  0.01  %
```

> Country which has the highest percentage of the people that earn >50k >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In [37]:
```
dfg2 = df.groupby(['native-country','salary'])['salary'].count().unstack()

dfg2['>50K perc'] = round( dfg2['>50K'] / ( dfg2['>50K'] + dfg2['<=50K'] ) * 100 , 2)

print('Country which has the highest percentage of the people that earn >50k  : \n '  )

dfg2[ dfg2['>50K perc'] == dfg2['>50K perc'].max() ]
```

```
Country which has the highest percentage of the people that earn >50k  :
```

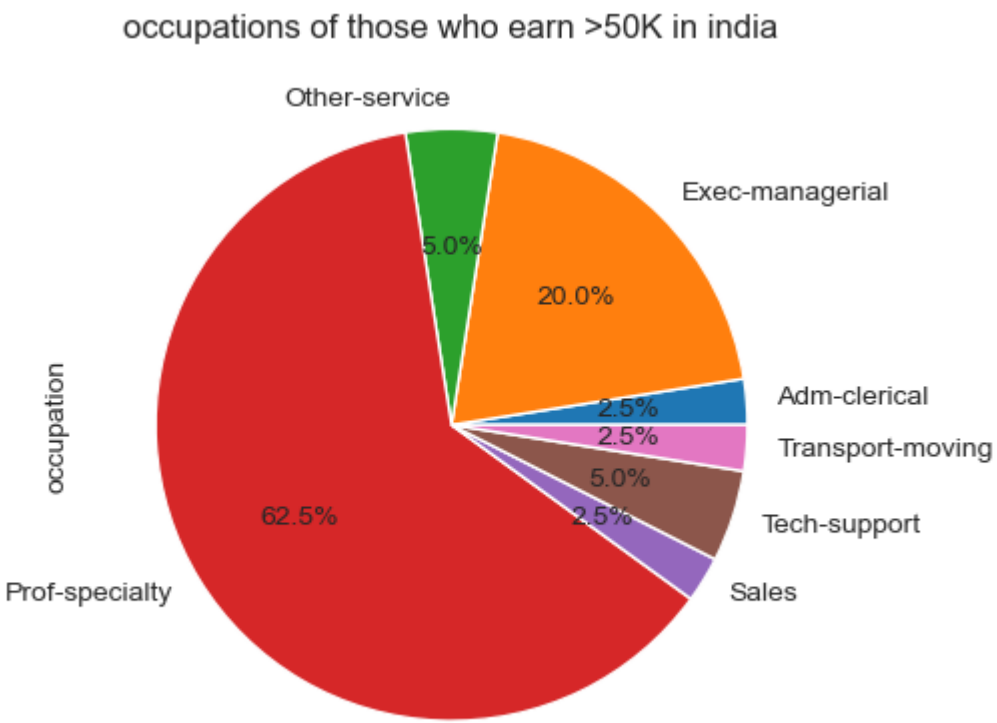
Out[37]:

salary	<=50K	>50K	>50K perc
native-country			
Iran	25.0	18.0	41.86

> Most popular occupation for those who earn >50K in india >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

In [50]:
```
dfg3 = df[
df['native-country'].isin( ['India'] ) & ( df['salary'] == '>50K' )
].groupby('occupation')['occupation'].count()

print('\n', dfg3 , '\n')

dfg3.plot.pie(
    autopct = '%1.1f%%'
)

plt.title('occupations of those who earn >50K in india')
plt.show()
```

```
 occupation
Adm-clerical         1
Exec-managerial      8
Other-service        2
Prof-specialty      25
Sales                1
Tech-support         2
Transport-moving     1
Name: occupation, dtype: int64
```



In [39]:
```
df # preview of the dataset
```

Out[39]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	nati coun
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	Unit Sta
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	Unit Sta
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	Unit Sta
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	Unit Sta
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cu
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	Unit Sta
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	Unit Sta
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	Unit Sta
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	Unit Sta
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	Unit Sta

32561 rows × 15 columns

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df=pd.read_csv('medical_examination.csv')
df.head()
```

Out[1]:

	id	age	sex	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

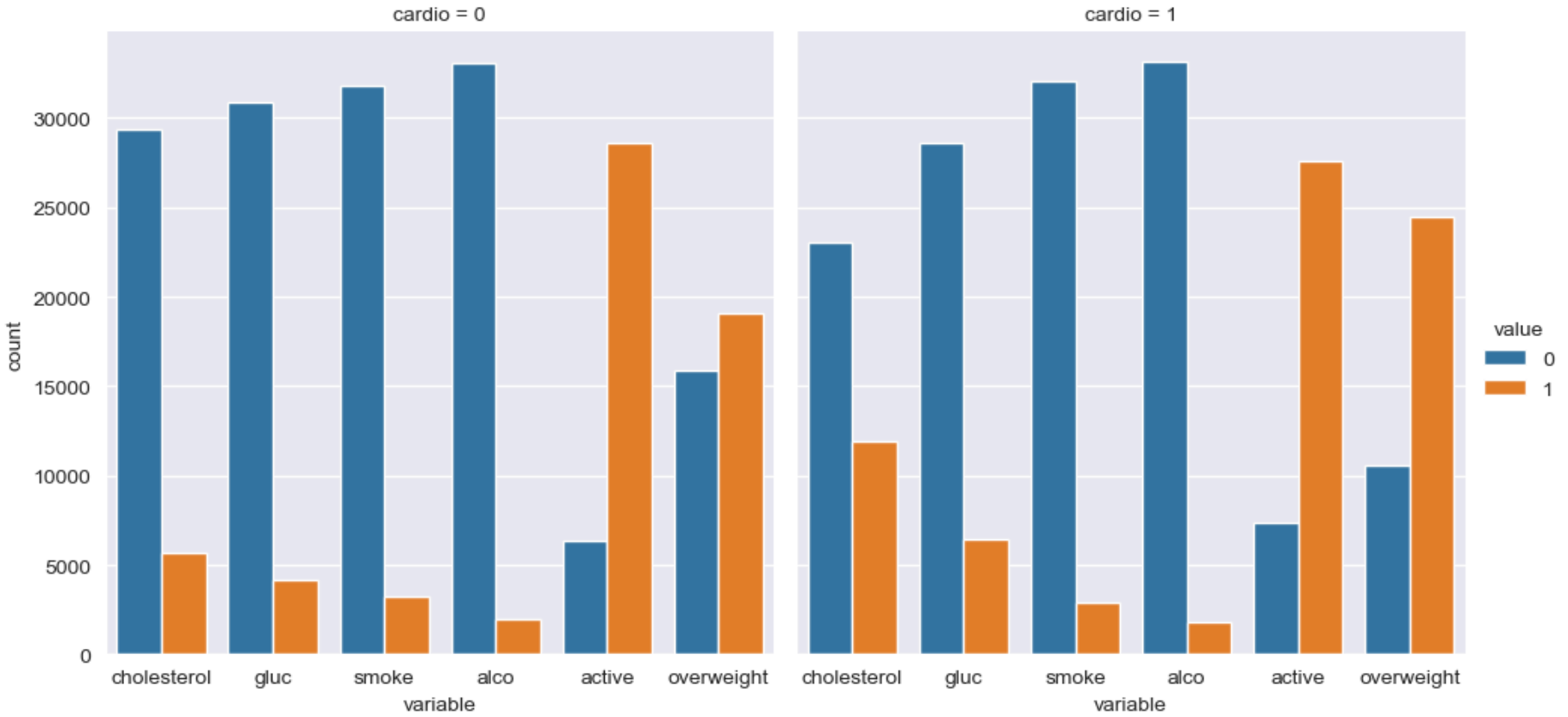
In [2]:
```
# firstly, need to calculate BMI using height (m) and weight (kg)
df['bmi']= df['weight'] / ( 0.0001 * df['height'] ** 2  )
df['overweight'] = df['bmi'].apply( lambda x : 1 if x > 25.0 else 0 )
```

In [3]:
```
# 0 always good , 1 is bad
df['cholesterol'] = df['cholesterol'].apply( lambda x : 0 if x == 1 else 1 )
df['gluc'] = df['gluc'].apply( lambda x : 0 if x == 1 else 1 )
```

In [4]:
```
df_cat = pd.melt( df , id_vars = ['cardio'] , value_vars = ['cholesterol','gluc','smoke','alco','active','overweight'] )
```

In [5]:
```
sns.set_style('darkgrid')
sns.catplot(
    data=df_cat,
    x='variable',
    hue='value',  # seperating the counts for each 0 and 1 value
    col='cardio', # seperating the plots according to the cardio value
    kind='count'
```

```
)
plt.show()
```

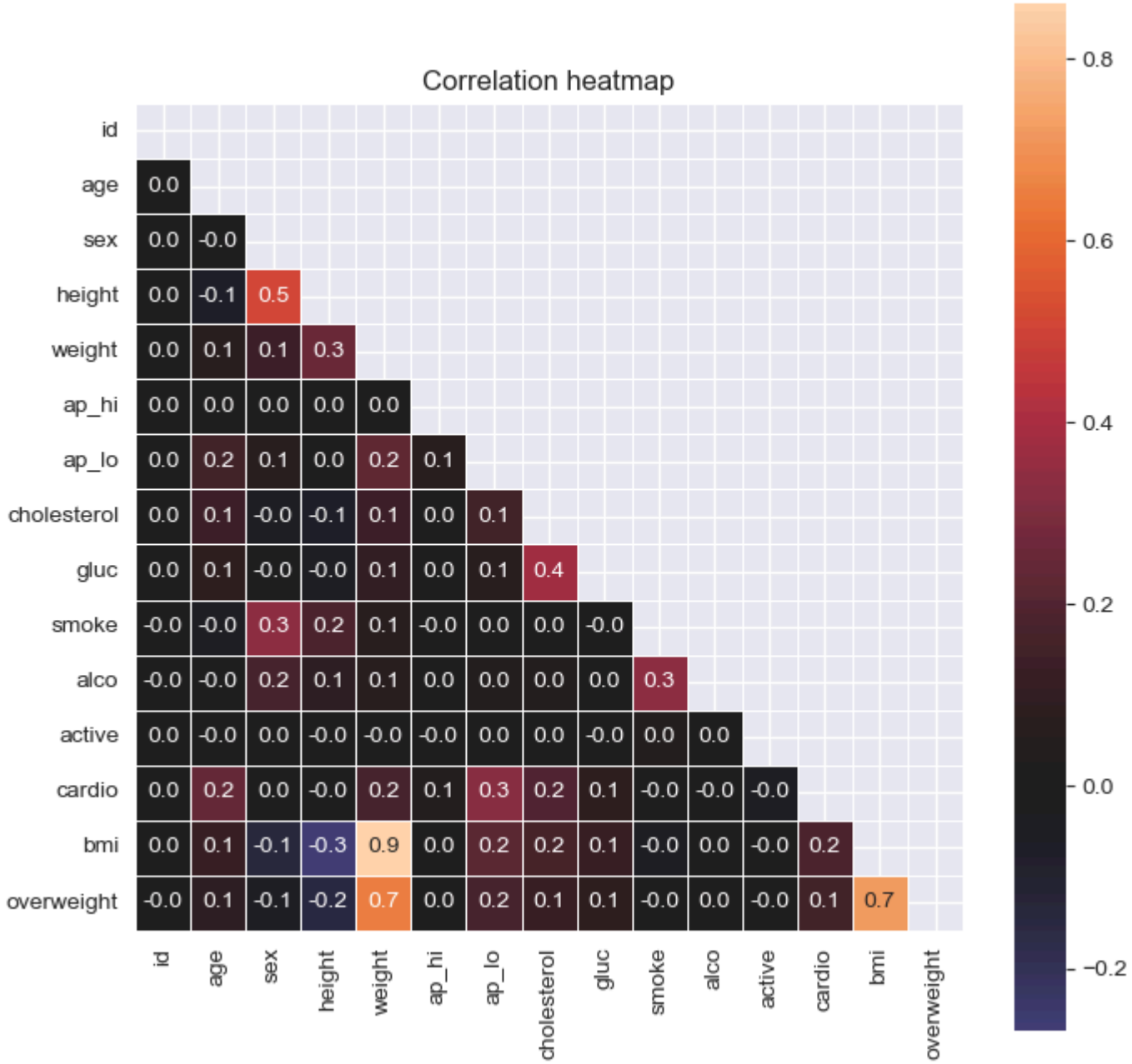


In [10]:
```
# data cleaning

df_heat = df[( df['ap_lo'] <= df['ap_hi'] )]

# removing 1st and last 2.5 % of data in the dataset

df_heat = df_heat[
( df_heat['height'] >= df_heat['height'].quantile(0.025))
& ( df_heat['height'] <= df_heat['height'].quantile(0.975))
& ( df_heat['weight'] >= df_heat['weight'].quantile(0.025))
```

```
& ( df_heat['weight'] <= df_heat['weight'].quantile(0.975))
]
```

In [11]:

```
corr = df_heat.corr() # making the correlation matrix
mask = np.triu(corr) # triu for upper and tril for lower triangle matrix
plt.figure(figsize=(8,8))
sns.heatmap(
    corr,
    mask = mask,
    fmt = '.1f',
    square = True,
    center=0,
    linewidths=0.5,
    annot = True, # displaying the values of the correlation
)
plt.title('Correlation heatmap')
plt.show()
```


Correlation heatmap
id
age 0.0
sex 0.0 -0.0
height 0.0 -0.1 0.5
weight 0.0 0.1 0.1 0.3
ap_hi 0.0 0.0 0.0 0.0 0.0
ap_lo 0.0 0.2 0.1 0.0 0.2 0.1
cholesterol 0.0 0.1 -0.0 -0.1 0.1 0.0 0.1
gluc 0.0 0.1 -0.0 -0.0 0.1 0.0 0.1 0.4
smoke -0.0 -0.0 0.3 0.2 0.1 -0.0 0.0 0.0 -0.0
alco -0.0 -0.0 0.2 0.1 0.1 0.0 0.0 0.0 0.0 0.3
active 0.0 -0.0 0.0 -0.0 -0.0 -0.0 0.0 0.0 -0.0 0.0 0.0
cardio 0.0 0.2 0.0 -0.0 0.2 0.1 0.3 0.2 0.1 -0.0 -0.0 -0.0
bmi 0.0 0.1 -0.1 -0.3 0.9 0.0 0.2 0.2 0.1 -0.0 0.0 -0.0 0.2
overweight -0.0 0.1 -0.1 -0.2 0.7 0.0 0.2 0.1 0.1 -0.0 0.0 -0.0 0.1 0.7
id age sex height weight ap_hi ap_lo cholesterol gluc smoke alco active cardio bmi overweight
0.8
0.6
0.4
0.2
0.0
−0.2

In [12]:
```
df
```

Out[12]:

	id	age	sex	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	bmi	overweight
0	0	18393	2	168	62.0	110	80	0	0	0	0	1	0	21.967120	0
1	1	20228	1	156	85.0	140	90	1	0	0	0	1	1	34.927679	1
2	2	18857	1	165	64.0	130	70	1	0	0	0	0	1	23.507805	0
3	3	17623	2	169	82.0	150	100	0	0	0	0	1	1	28.710479	1
4	4	17474	1	156	56.0	100	60	0	0	0	0	0	0	23.011177	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	99993	19240	2	168	76.0	120	80	0	0	1	0	1	0	26.927438	1
69996	99995	22601	1	158	126.0	140	90	1	1	0	0	1	1	50.472681	1
69997	99996	19066	2	183	105.0	180	90	1	0	0	1	0	1	31.353579	1
69998	99998	22431	1	163	72.0	135	80	0	1	0	0	0	1	27.099251	1
69999	99999	20540	1	170	72.0	120	80	1	0	0	0	1	0	24.913495	0

70000 rows × 15 columns

In [254…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('fcc-forum-pageviews.csv', index_col = 'date')
# date column will be the index column

col = ['orangered','navy','chartreuse','darkviolet','tomato','tan','purple','coral','green','deeppink','red','skyblue']

df.head()
```

Out[254…

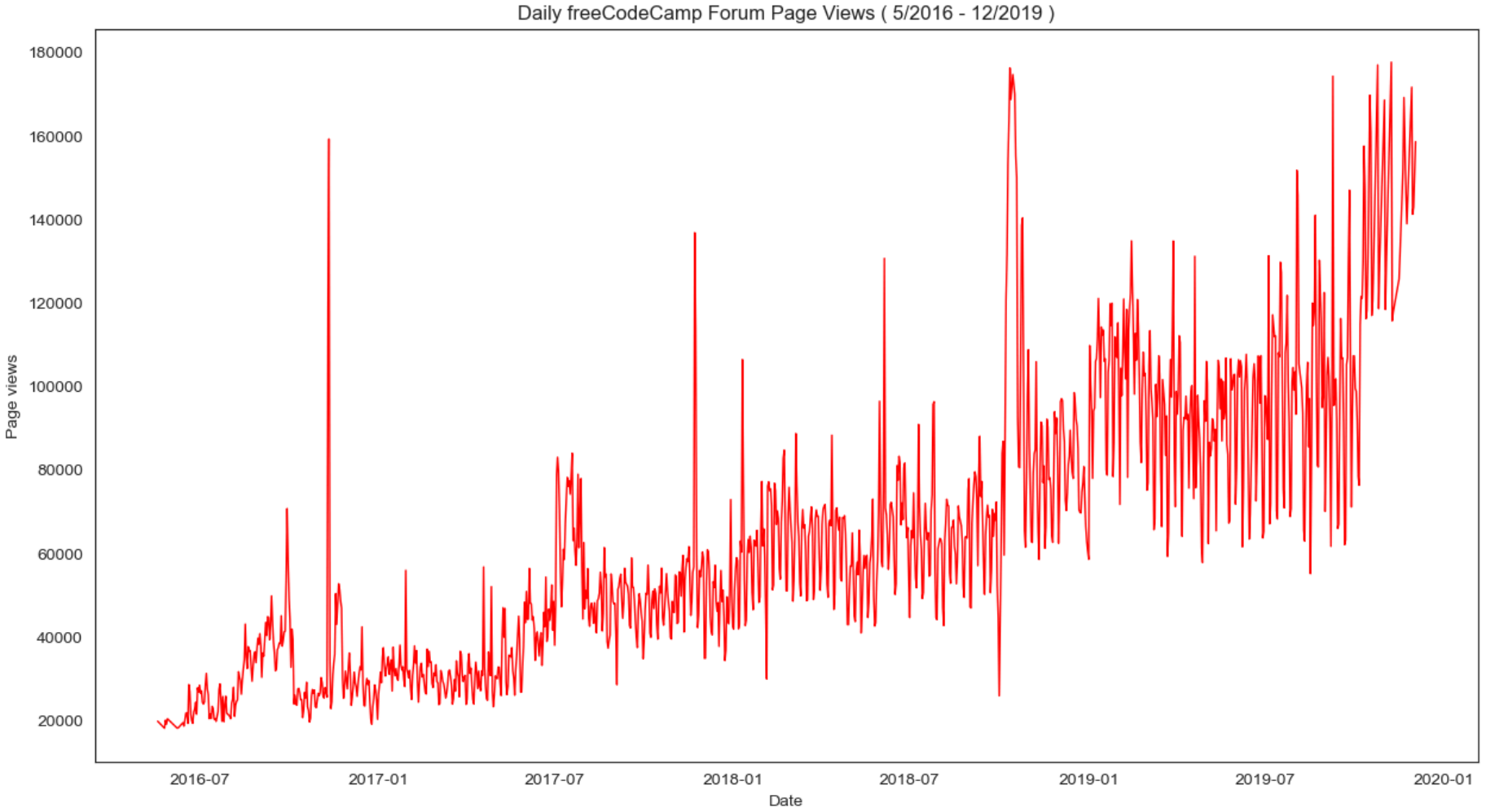
date	value
2016-05-09	1201
2016-05-10	2329
2016-05-11	1716
2016-05-12	10539
2016-05-13	6933

In [255…
```
# date_parsed column will display the date as 'datetime' datat type
# bcz origional date column is displayed as 'object' data type

df['date_parsed'] = pd.to_datetime(df.index , format = 'mixed')
df['year'] = df['date_parsed'].dt.year
df['month'] = df['date_parsed'].dt.month
# dt.strftime(' %b ') for month name like 'jan'
# %B for month name like 'january'

# removing top 2.5% and bottom 2.5%  of data from the dataset

df = df[
        (df['value'] >= df['value'].quantile(0.025)) &
```

```
        (df['value'] <= df['value'].quantile(0.975))
    ]
```

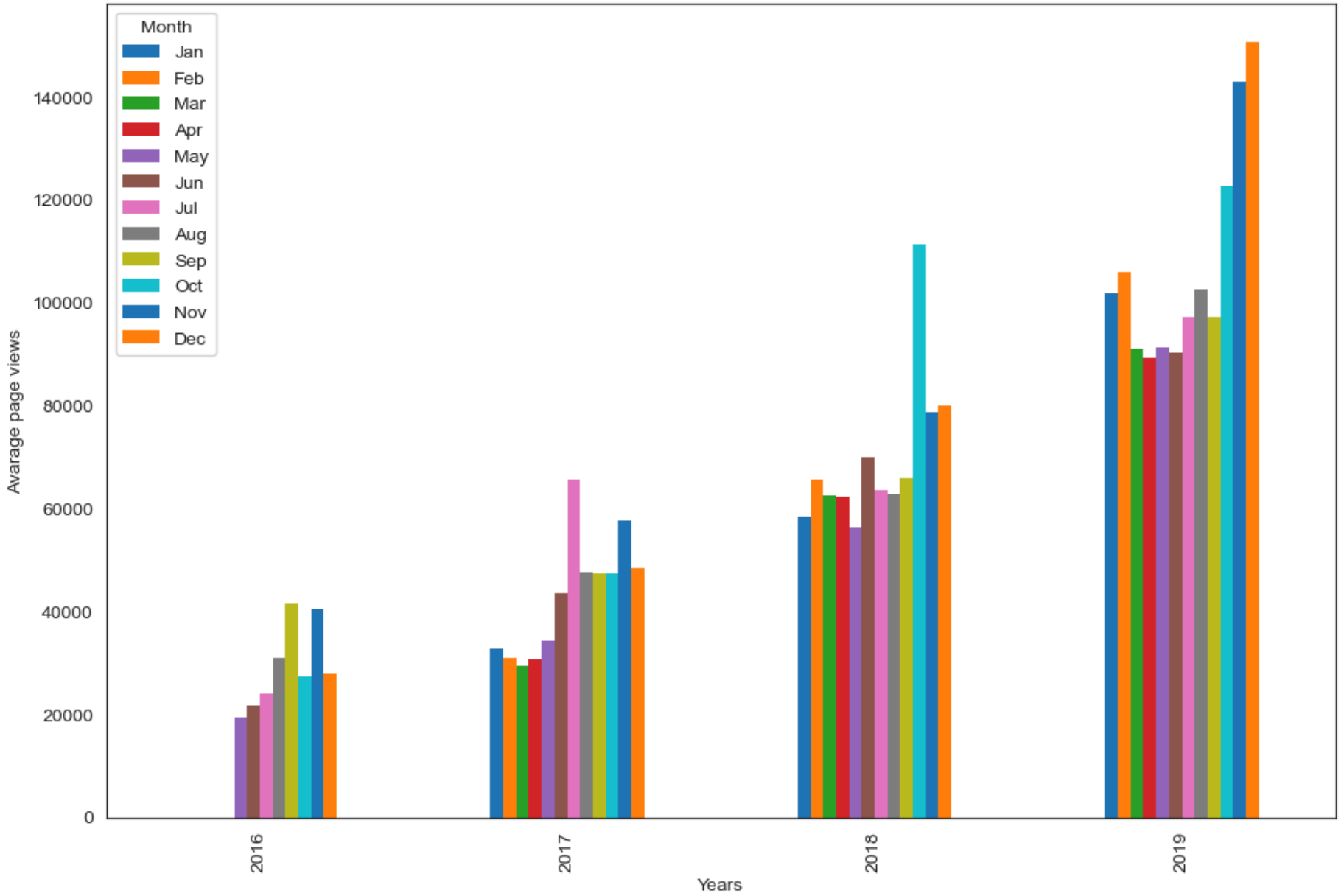
In [256…

```
sns.set_style('white')
plt.figure( figsize = (15,8) )
sns.lineplot(
    x = df['date_parsed'],
    y = df['value'],
    linewidth=1,
    color = 'red'
)
plt.title('Daily freeCodeCamp Forum Page Views ( 5/2016 - 12/2019 )')
plt.xlabel('Date')
plt.ylabel('Page views')
plt.show()
```



In [257…
```
dfg1 = df.groupby(['year','month'])['value'].mean().unstack()
mon_names = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
```

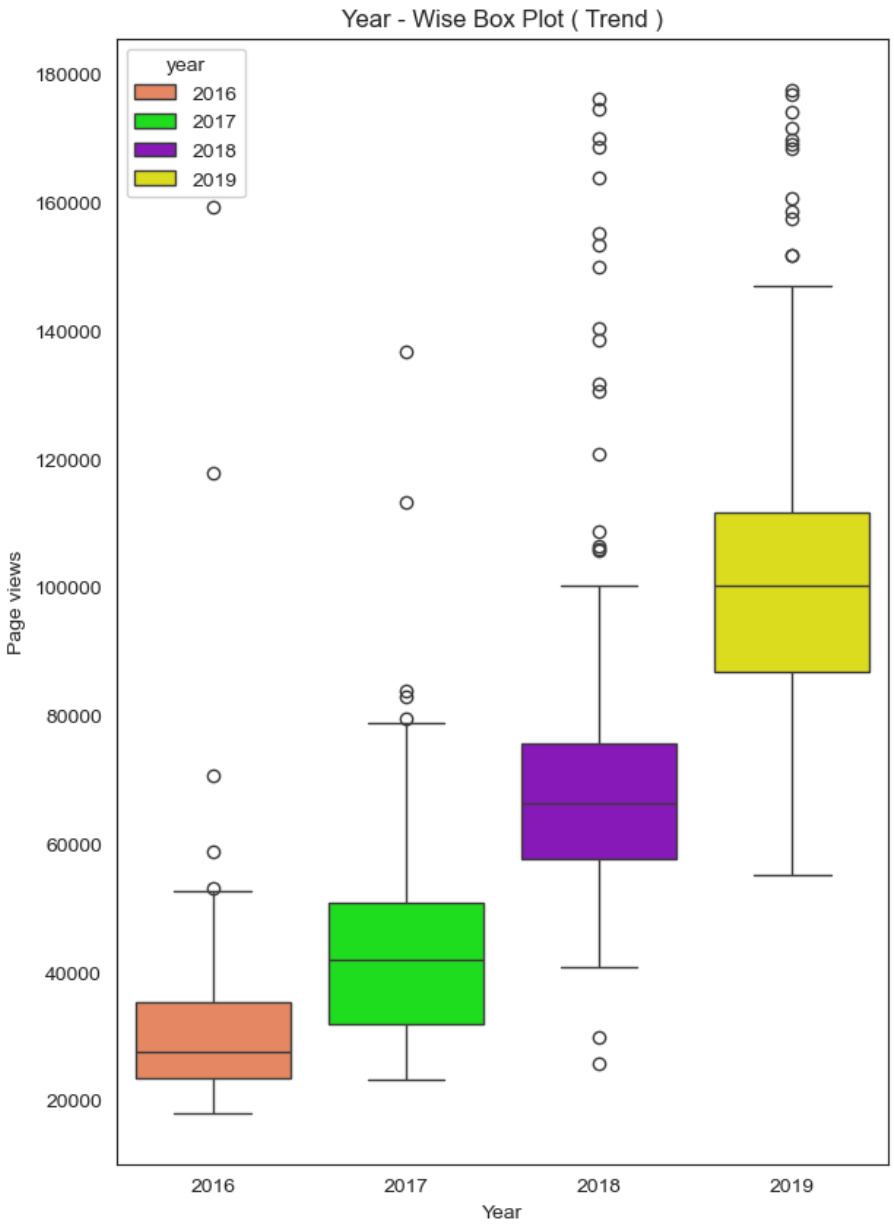
In [258…
```
dfg1.plot.bar( linewidth=0 , figsize = (12,8) )
plt.xlabel('Years')
plt.ylabel('Avarage page views')
plt.legend(title = 'Month' , labels = mon_names)
plt.show()
```

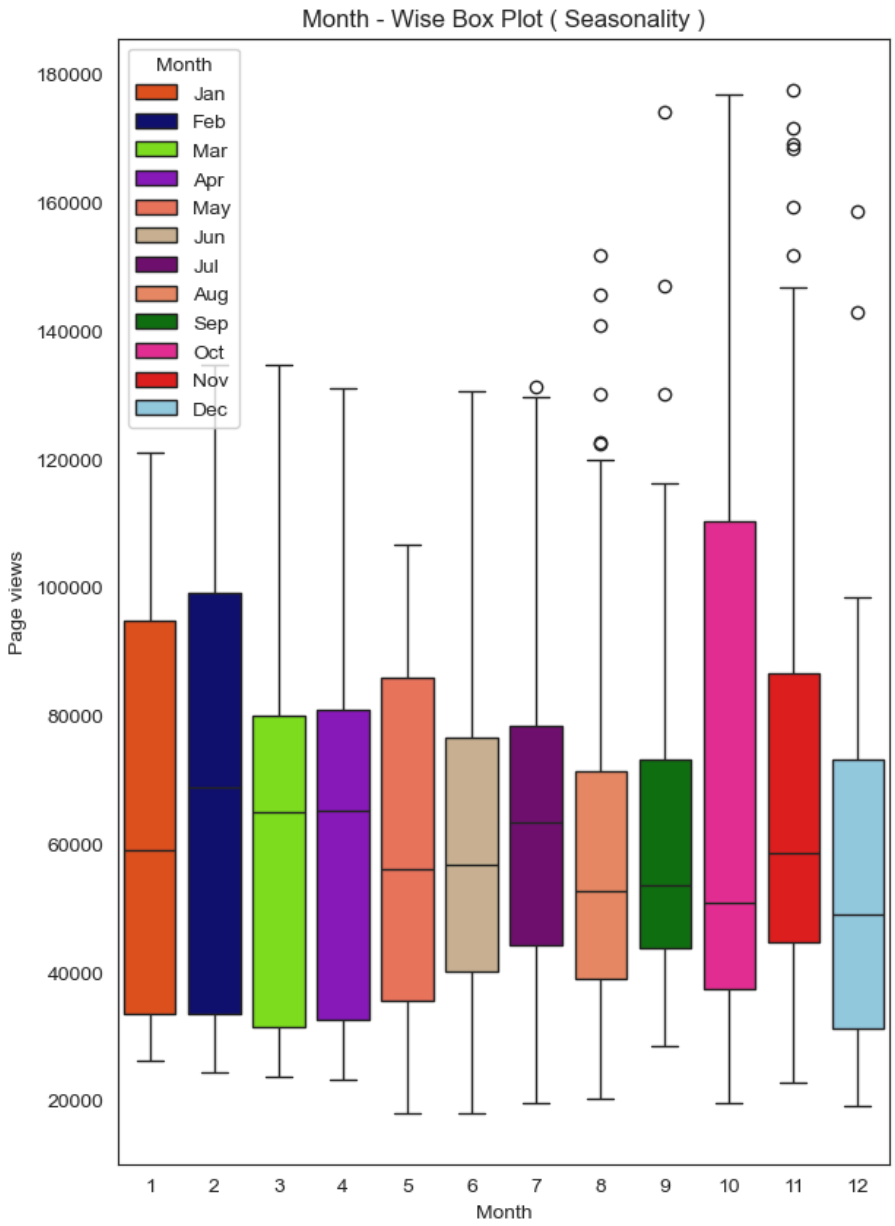


In [259…

```
plt.figure(figsize=(15,10))

plt.subplot(1,2,1)
```

```
sns.boxplot(
    x = df.year ,
    y = df.value ,
    palette = ['coral','lime','darkviolet','yellow'] ,
    hue = df.year
    )
plt.title('Year - Wise Box Plot ( Trend )')
plt.xlabel('Year')
plt.ylabel('Page views')


plt.subplot(1,2,2)
sns.boxplot(
    x = df.month ,
    y = df.value ,
    palette = col ,
    hue = df.month
    )
plt.title('Month - Wise Box Plot ( Seasonality )')
plt.xlabel('Month')
plt.ylabel('Page views')
plt.legend(title = 'Month' , labels = mon_names)

plt.show()
```


Year - Wise Box Plot (Trend)
year
2016
2017
2018
2019
Page views
Year
180000
160000
140000
120000
100000
80000
60000
40000
20000
2016
2017
2018
2019


Month - Wise Box Plot (Seasonality)
Month
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Page views
Month
180000
160000
140000
120000
100000
80000
60000
40000
20000
1
2
3
4
5
6
7
8
9
10
11
12

In [260…

```
df
```

Out[260...

date	value	date_parsed	year	month
2016-05-19	19736	2016-05-19	2016	5
2016-05-26	18060	2016-05-26	2016	5
2016-05-27	19997	2016-05-27	2016	5
2016-05-28	19044	2016-05-28	2016	5
2016-05-29	20325	2016-05-29	2016	5
...	...	...	...	...
2019-11-24	138875	2019-11-24	2019	11
2019-11-29	171584	2019-11-29	2019	11
2019-11-30	141161	2019-11-30	2019	11
2019-12-01	142918	2019-12-01	2019	12
2019-12-03	158549	2019-12-03	2019	12

1238 rows × 4 columns

In [7]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as sc

df = pd.read_csv('epa-sea-level.csv')
df.head()
```

Out[7]:

	Year	CSIRO Adjusted Sea Level	Lower Error Bound	Upper Error Bound	NOAA Adjusted Sea Level
0	1880	0.000000	-0.952756	0.952756	NaN
1	1881	0.220472	-0.732283	1.173228	NaN
2	1882	-0.440945	-1.346457	0.464567	NaN
3	1883	-0.232283	-1.129921	0.665354	NaN
4	1884	0.590551	-0.283465	1.464567	NaN

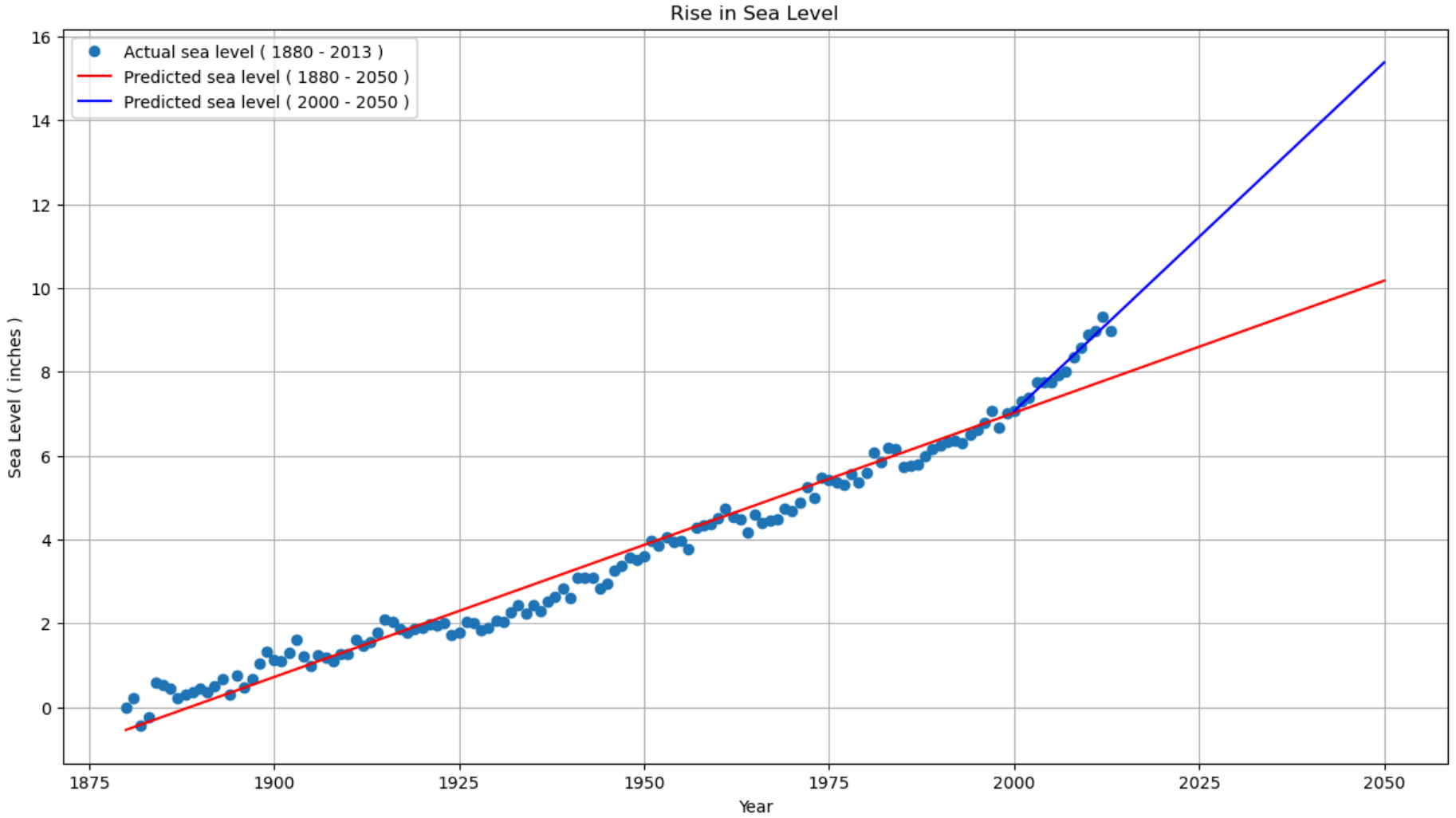
In [66]:
```
# Regression Model 1 : from 1880 to 2013

reg_model1 = sc.linregress(df['Year'],df['CSIRO Adjusted Sea Level'])
b01 = reg_model1.intercept
b11 = reg_model1.slope


# Regression Model 2 : from 2000 to 2013

df1 = df[ df['Year'] >= 2000 ]
reg_model1 = sc.linregress(df1['Year'],df1['CSIRO Adjusted Sea Level'])
b02 = reg_model1.intercept
b12 = reg_model1.slope


print('\n Regression Model 1 : from 1880 to 2013 > The slope is : ', b11 , ' and The intercept is : ', b01 ,'\n')
print(' Regression Model 2 : from 2000 to 2013 > The slope is : ', b12 , ' and The intercept is : ', b02 ,'\n')
```

```
def reg_func(x,slope,intercept):  # defining regression function
    return intercept + slope * x
```

```
Regression Model 1 : from 1880 to 2013 > The slope is :  0.0630445840121348  and The intercept is :  -119.06594196773978

Regression Model 2 : from 2000 to 2013 > The slope is :  0.1664272733318682  and The intercept is :  -325.7934668059649
```

In [72]:

```
# scatter plot
plt.figure(figsize=(15,8))
plt.plot(
    df['Year'] ,
    df['CSIRO Adjusted Sea Level'] ,
    'o'  # 'o' for o symbols
        )

# predictions
# from 1880 to 2013
x1 = range(1880,2051)
plt.plot( x1 , reg_func(x1,b11,b01) , 'r')

# from 2000 to 2050
x2 = range(2000,2051)
plt.plot( x2 , reg_func(x2,b12,b02) , 'b')

plt.title('Rise in Sea Level')
plt.ylabel('Sea Level ( inches )')
plt.xlabel('Year')
plt.grid(True)
plt.legend(['Actual sea level ( 1880 - 2013 )' , 'Predicted sea level ( 1880 - 2050 )' , 'Predicted sea level ( 2000 - 2050 )'

plt.show()
```


Rise in Sea Level
Actual sea level (1880 - 2013)
Predicted sea level (1880 - 2050)
Predicted sea level (2000 - 2050)
Sea Level (inches)
Year
16
14
12
10
8
6
4
2
0
1875
1900
1925
1950
1975
2000
2025
2050

In [75]:
```
df
```

Out[75]:

	Year	CSIRO Adjusted Sea Level	Lower Error Bound	Upper Error Bound	NOAA Adjusted Sea Level
0	1880	0.000000	-0.952756	0.952756	NaN
1	1881	0.220472	-0.732283	1.173228	NaN
2	1882	-0.440945	-1.346457	0.464567	NaN
3	1883	-0.232283	-1.129921	0.665354	NaN
4	1884	0.590551	-0.283465	1.464567	NaN
...	...	...	...	...	...
129	2009	8.586614	8.311024	8.862205	8.046354
130	2010	8.901575	8.618110	9.185039	8.122973
131	2011	8.964567	8.661417	9.267717	8.053065
132	2012	9.326772	8.992126	9.661417	8.457058
133	2013	8.980315	8.622047	9.338583	8.546648

134 rows × 5 columns