Analyzing Birth rate from World Bank Data using Agglomerative Clustering and Logistic Regression

In [1]:

```
import pandas as pd
import numpy as np
import missingno as ms
import scipy.optimize as opt
import sklearn.cluster as cluster

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

Dataset Loading

In [2]:

```
def function_transpose(file):
    '''
    this function will take a .csv file in the world bank format and transpose it in
    format
    '''
    dataset=pd.read_csv(file)
    dataset=dataset.transpose()
    dataset.columns=dataset.iloc[0]
    dataset=dataset.iloc[:-1]
    dataset=dataset.reset_index()
    dataset=dataset.rename(columns={"index": "Year"})
    #dataset=dataset.fillna(0)

    return dataset

file = 'countriesOfTheWorld-WDI.csv'
function_transpose(file)
```

Out[2]:

Country	Year	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Anguilla
0	Country	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Anguilla
1	Region	ASIA (EX. NEAR EAST)	EASTERN EUROPE	NORTHERN AFRICA	OCEANIA	WESTERN EUROPE	SUB-SAHARAN AFRICA	LATIN AMER. & CARIB
2	Population	31056997	3581655	32930091	57794	71201	12127071	13477
3	Area (sq. mi.)	647500	28748	2381740	199	468	1246700	102

Country	Year	Afghanistan	Albania	Algeria	American Samoa	Andorra	Angola	Anguilla
4	Pop. Density (per sq. mi.)	48,0	124,6	13,8	290,4	152,1	9,7	132,1
5	Coastline (coast/area ratio)	0,00	1,26	0,04	58,29	0,00	0,13	59,80
6	Net migration	23,06	-4,93	-0,39	-20,71	6,6	0	10,76
7	Infant mortality (per 1000 births)	163,07	21,52	31	9,27	4,05	191,19	21,03
8	GDP ($ per capita)	700.0	4500.0	6000.0	8000.0	19000.0	1900.0	8600.0
9	Literacy (%)	36,0	86,5	70,0	97,0	100,0	42,0	95,0
10	Phones (per 1000)	3,2	71,2	78,1	259,5	497,2	7,8	460,0
11	Arable (%)	12,13	21,09	3,22	10	2,22	2,41	0
12	Crops (%)	0,22	4,42	0,25	15	0	0,24	0
13	Other (%)	87,65	74,49	96,53	75	97,78	97,35	100
14	Climate	1	3	1	2	3	NaN	2
15	Birthrate	46,6	15,11	17,14	22,46	8,71	45,11	14,17
16	Deathrate	20,34	5,22	4,61	3,27	6,25	24,2	5,34
17	Agriculture	0,38	0,232	0,101	NaN	NaN	0,096	0,04
18	Industry	0,24	0,188	0,6	NaN	NaN	0,658	0,18

19 rows × 228 columns

In [3]:

```
dataset = pd.read_csv('countriesOfTheWorld-WDI.csv')
```

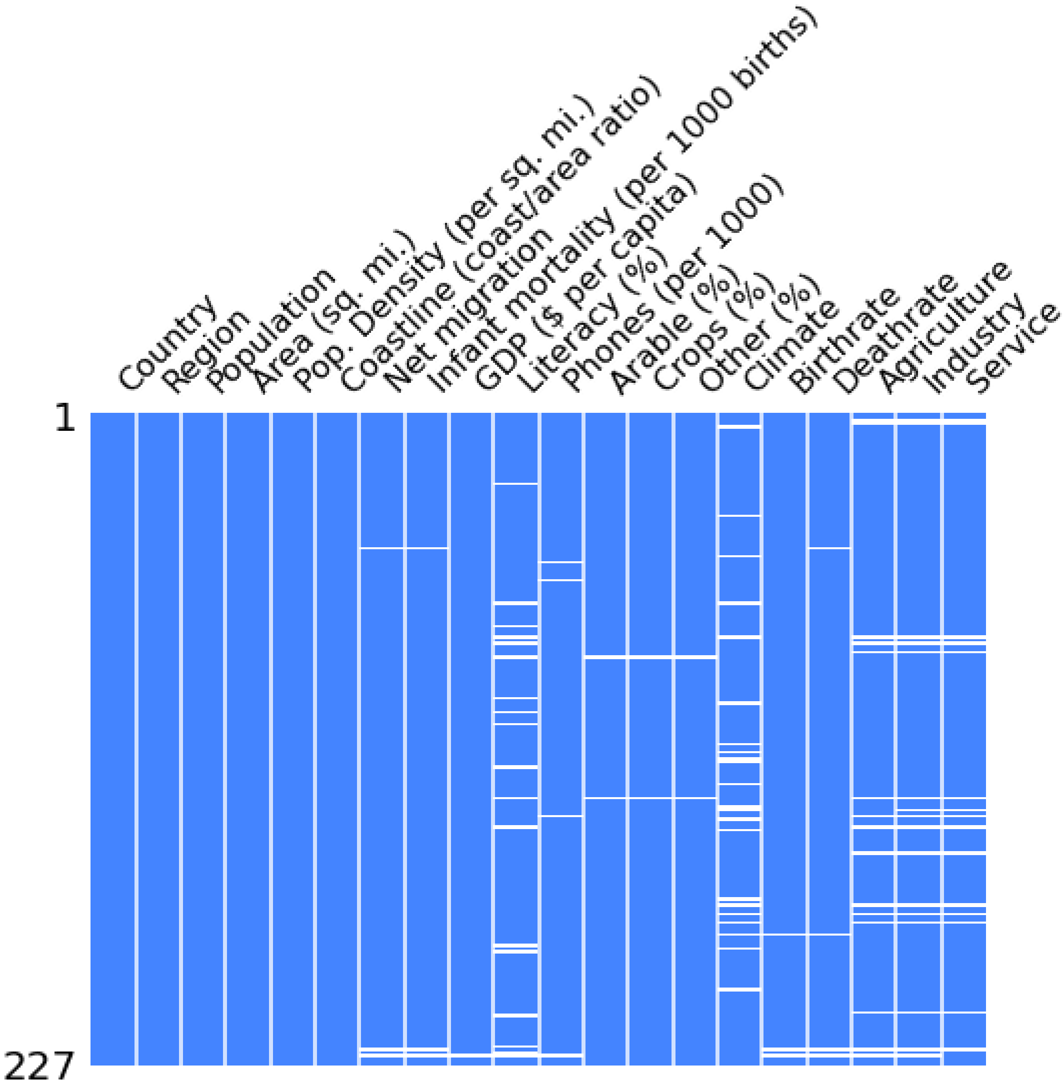
Data Pre-processing

Lets see the missing values first and then fix those outliers. This time I am using a library names "missingno", It will show the missing values in a viusal effects. We will then analyze the missing values in each features and then try to fix those.

In [4]:

```
fig, ax = plt.subplots(figsize=(8,6))
ms.matrix(dataset, ax=ax, sparkline=False, color=(0.27, 0.52, 1.0))
plt.show
```

Out[4]: <function matplotlib.pyplot.show(close=None, block=None)>



The matrix plot is showing the missing values trend in every feature. As it is visible that some of the features, For example Agriculture, Industry and Service; are showing missing values in the same country (the straight line). Let fill these missing feature values with the mean value of the respective column.

In [5]:

```
dataset.fillna(dataset.mean(), inplace=True)
```

In [6]:

```
dataset.isnull().sum()
```

Out[6]:

```
Country                               0
Region                                0
Population                            0
Area (sq. mi.)                        0
Pop. Density (per sq. mi.)            0
Coastline (coast/area ratio)          0
Net migration                         3
Infant mortality (per 1000 births)    3
GDP ($ per capita)                    0
Literacy (%)                         18
Phones (per 1000)                     4
Arable (%)                            2
Crops (%)                             2
Other (%)                             2
Climate                              22
```

```
Birthrate                          3
Deathrate                          4
Agriculture                       15
Industry                          16
Service                           15
dtype: int64
```

In [7]:

```
columns = dataset[['Net migration', 'Deathrate', 'Agriculture', 'Industry', 'Service
                   'Infant mortality (per 1000 births)', 'Literacy (%)', 'Phones (pe
                   'Arable (%)', 'Crops (%)', 'Other (%)', 'Climate', 'Birthrate']]
def changetype(columns):
    '''
    This function is used in the coversion of the feature column types as
    some are objects and some are floats. And replace all the , with the
    . in all numeric features for smooth visualizations.
    '''
    for i in columns:
        dataset[i] = dataset[i].astype(str)
        dataset1 = []
        for j in dataset[i]:
            j = j.replace(',','.')
            j = float(j)
            dataset1.append(j)
        dataset[i] = dataset1
changetype(columns)
```

In [8]:

```
# trim the spaces after and before the text. It can be seen from the sample
# data, there are some spaces in some countries names.

dataset['Region'] = dataset.Region.str.strip()
dataset['Country'] = dataset.Country.str.strip()
```

In [9]:

```
df1 = dataset.copy() # copy dataset in df1
df2 = df1.drop(columns= ['Pop. Density (per sq. mi.)', 'Coastline (coast/area ratio)
               inplace = True ) # Drop the column Region as we will analysis the dat
```

Dataset Normalization

In [10]:

```
def norm(array):
    """ Returns array normalised to [0,1]. Array can be a numpy array
    or a column of a dataframe"""

    min_val = np.min(array)
    max_val = np.max(array)

    scaled = (array-min_val) / (max_val-min_val)

    return scaled


def norm_df(df):
    """
    Returns all columns of the dataframe normalised to [0,1] with the
    exception the first (containing the names)
    Calls function norm to do the normalisation of one column, but
    doing all in one function is also fine.
    """
```

```
    # iterate over all columns
    for col in df.columns[2:]:      # excluding the first column
        df[col] = norm(df[col])
    return df
```

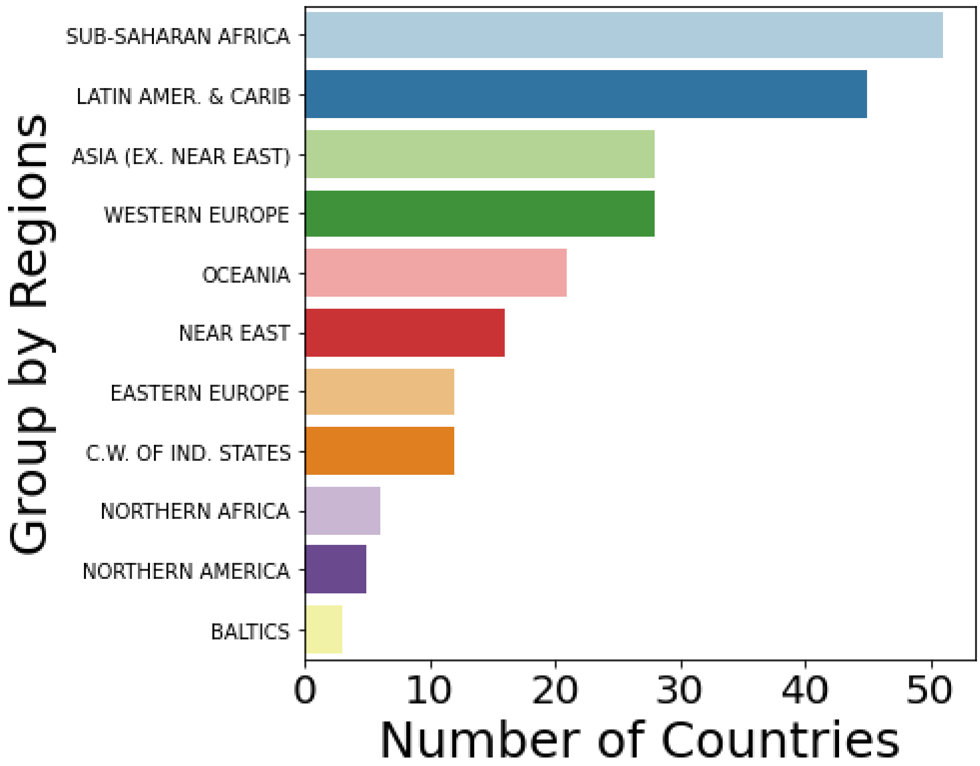
In [11]:

```
df1 = norm_df(df1)
df1=df1.fillna(0)
```

Number of countries in each region

In [12]:

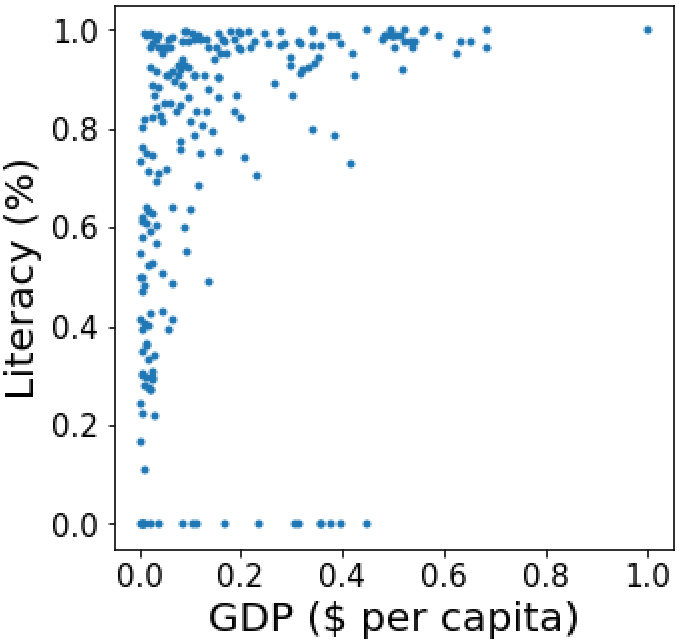
```
import seaborn as sns

country = df1['Region'].value_counts()
plt.figure(figsize=(6,6,))
sns.barplot(y=country.index,x=country.values, palette="Paired")
plt.xlabel('Number of Countries', fontsize=25)
plt.ylabel('Group by Regions', fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=10)
plt.show()
```

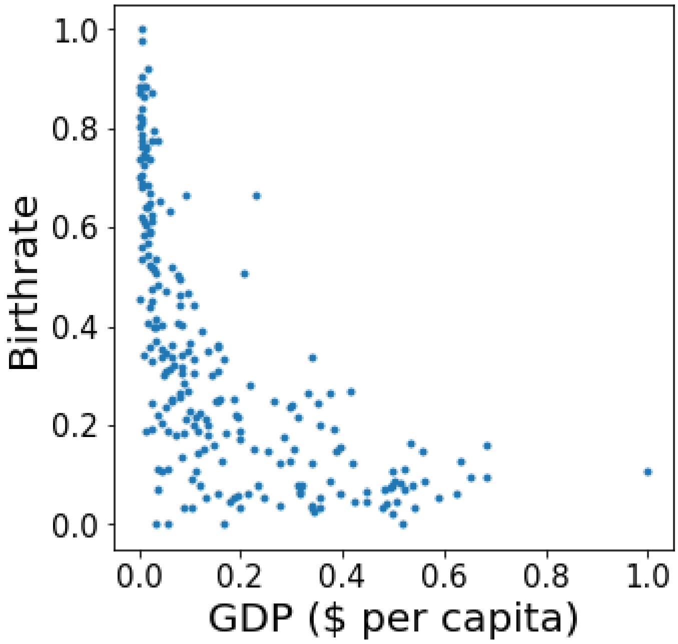


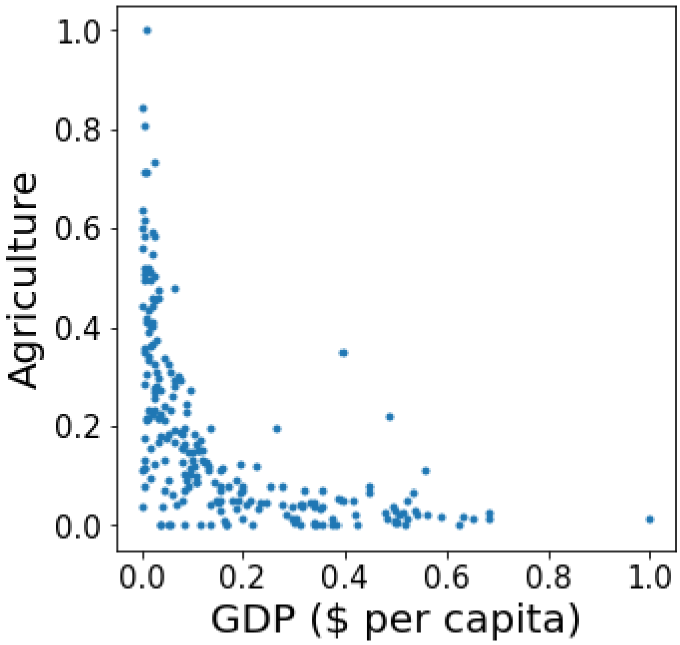
In [13]:

```
def makeplot(df, col1, col2):
    """
    Produces a square plot of two columns of dataframe df using small circle
    symbols.
    """

    plt.figure(figsize=(5.0,5.0))
    plt.plot(df[col1], df[col2], "o", markersize=3)

    plt.xlabel(col1,fontsize=20)
    plt.ylabel(col2,fontsize=20)
    plt.xticks(fontsize=15)
```

```
    plt.yticks(fontsize=15)
    plt.show()


# exploratory plots
makeplot(df1, "GDP ($ per capita)", "Literacy (%)")
makeplot(df1, "GDP ($ per capita)", "Birthrate")
makeplot(df1, "GDP ($ per capita)", "Agriculture")
```



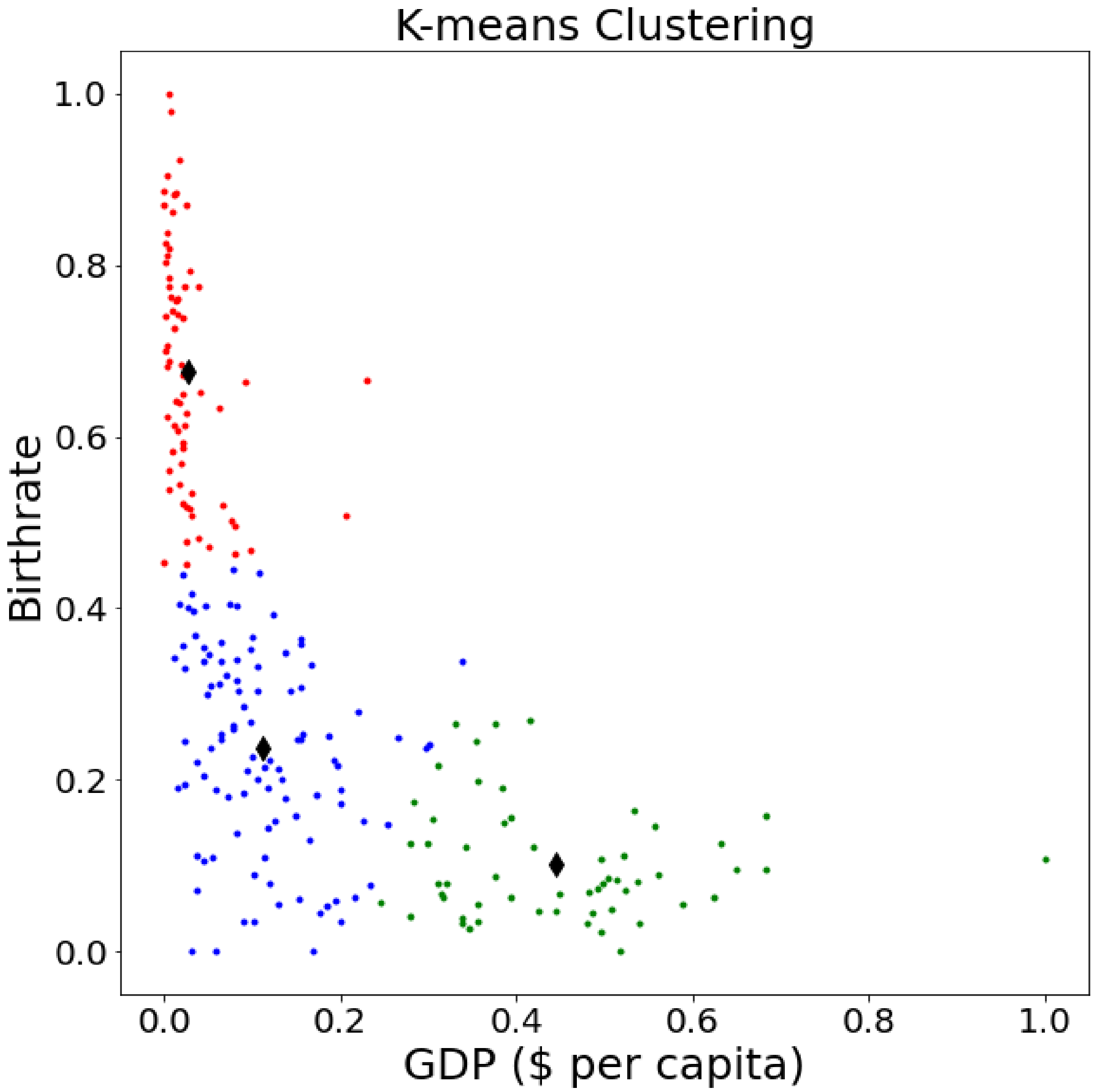




K-means Clustering

In [14]:

```
####### kmeans set up the clusterer, 4 expected clusters
kmeans = cluster.KMeans(n_clusters=3)

# extract columns for fitting
df_fit = df1[[ "GDP ($ per capita)", "Birthrate"]].copy()

kmeans.fit(df_fit)

# extract labels and cluster centres
labels = kmeans.labels_
cen = kmeans.cluster_centers_

# plot using the labels to select colour
plt.figure(figsize=(10,10))

col = ["blue", "red", "green", "magenta", "yellow", "red"]
for l in range(3):     # loop over the different labels
    plt.plot(df_fit["GDP ($ per capita)"][labels==l], df_fit["Birthrate"][labels==l]

# show cluster centres
for ic in range(3):
    xc, yc = cen[ic,:]
    plt.plot(xc, yc, "dk", markersize=10)

plt.title("K-means Clustering", fontsize=25)
plt.xlabel("GDP ($ per capita)", fontsize=25)
plt.ylabel("Birthrate", fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```

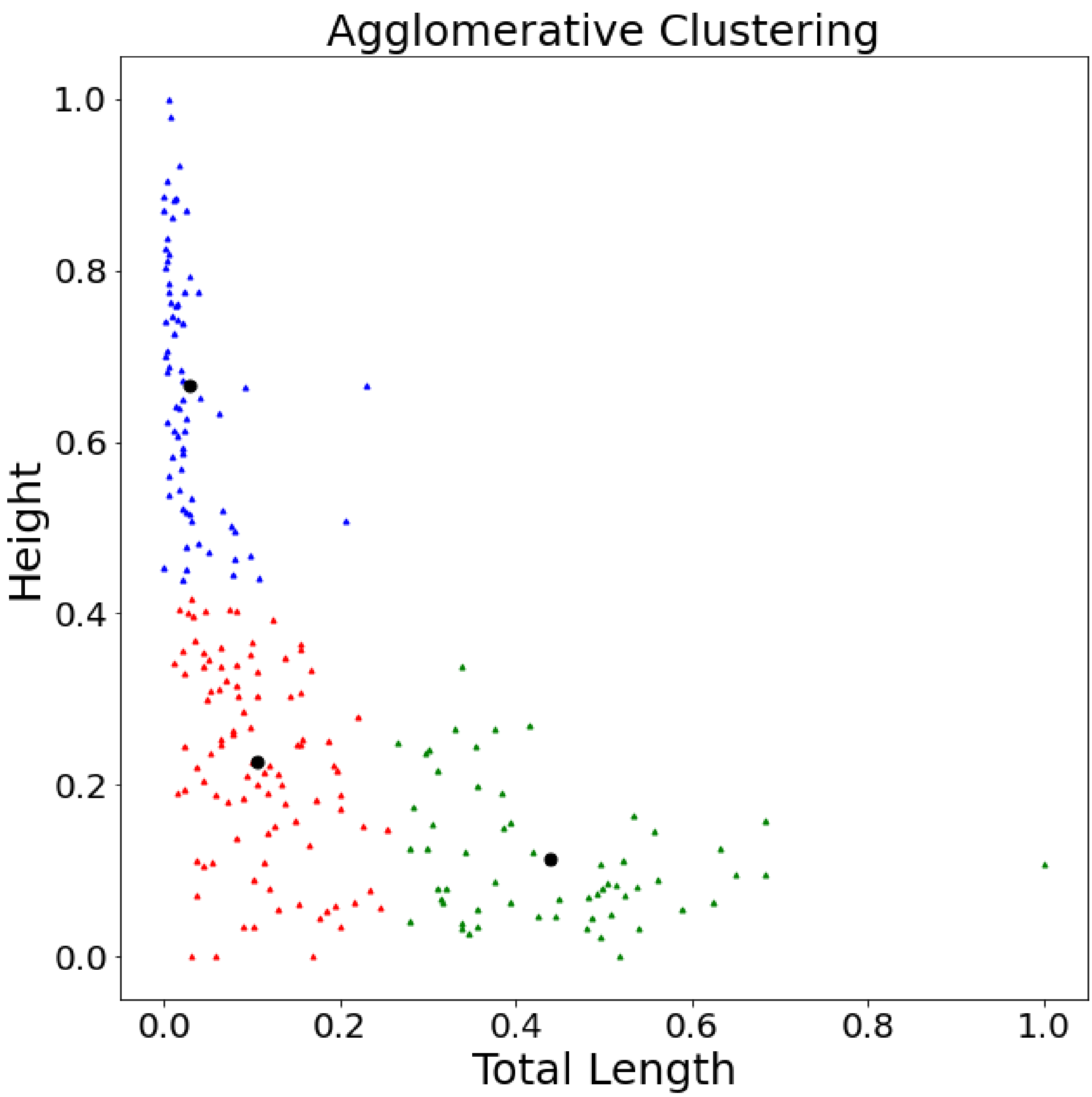


Agglomerative Clustering

In [15]:

```
##### setting up agglomerative clustering for 6 clusters
ac = cluster.AgglomerativeClustering(n_clusters=3)

# carry out the fitting
df_fit = df1[[ "GDP ($ per capita)", "Birthrate"]].copy()
ac.fit(df_fit)

labels = ac.labels_

# The clusterer does not return cluster centres, but they are easily computed
xcen = []
ycen = []
for ic in range(3):
    xc = np.average(df_fit["GDP ($ per capita)"][labels==ic])
    yc = np.average(df_fit["Birthrate"][labels==ic])
    xcen.append(xc)
    ycen.append(yc)

# plot using the labels to select colour
plt.figure(figsize=(10,10))
```

```
col = ["blue", "red", "green", "magenta","yellow","aqua"]
for l in range(0,3):     # loop over the different labels
    plt.plot(df_fit["GDP ($ per capita)"][labels==l], df_fit["Birthrate"][labels==l]

# show cluster centres
for ic in range(3):
    plt.plot(xcen[ic], ycen[ic], ".", markersize=14, color = "k")

plt.title("Agglomerative Clustering", fontsize=25)
plt.xlabel("Total Length", fontsize=25)
plt.ylabel("Height", fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()


###### writing labels into df_fish, sorting and exporting as excel file
df1["labels"] = labels
df1 = df1.sort_values(["labels"], ignore_index=True)
#df.to_excel("fish_clusters.xlsx")
```



Data Fitting

In [16]:
```
from sklearn.linear_model import LinearRegression
data = pd.read_csv("WDI_country.csv")
data
```

Out[16]:

	year	Population	Birthrate % Change
0	1950	376325200	NaN
1	1951	382376948	1.61
2	1952	388799073	1.68
3	1953	395544369	1.73
4	1954	402578596	1.78
...	...	...	...
63	2013	1280842125	1.19
64	2014	1295600772	1.15
65	2015	1310152403	1.12
66	2016	1324517249	1.10
67	2017	1338676785	1.07

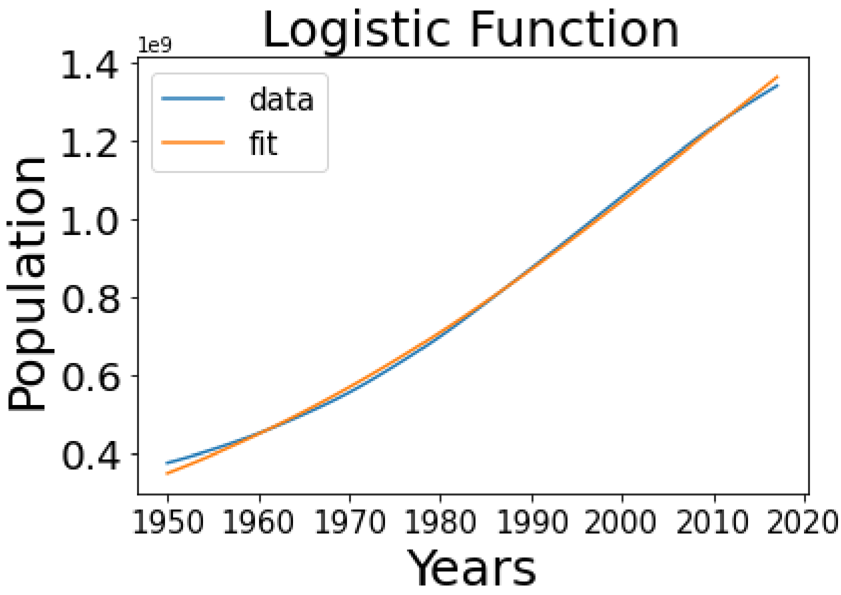
68 rows × 3 columns

In [17]:
```
"""
Define the logistics functions for fitting.
"""

def logistics(t, scale, growth, t0):
    """ Computes logistics function with scale, growth raat
    and time of the turning point as free parameters
    """

    f = scale / (1.0 + np.exp(-growth * (t - t0)))

    return f
```

In [18]:
```
"""
fit the logistics function with some initial parameters such as p0. It will give us
fit logistic growth and then calculate/ plot the result.
"""

# fit exponential growth
p, c = opt.curve_fit(logistics, data["year"], data["Population"], p0=(2e9, 0.05, 199
# much better
print("Fit parameter", p)

data["logistic"] = logistics(data["year"], *p)

plt.figure()
plt.plot(data["year"], data["Population"], label="data")
plt.plot(data["year"], data["logistic"], label="fit")

plt.legend(fontsize=15)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Population", fontsize=25)
```

```
plt.title("Logistic Function", fontsize=25)
plt.xticks(fontsize=15)
plt.yticks(fontsize=20)
plt.show()
print()
```

```
Fit parameter [2.52480676e+09 2.96170675e-02 2.01171142e+03]
```



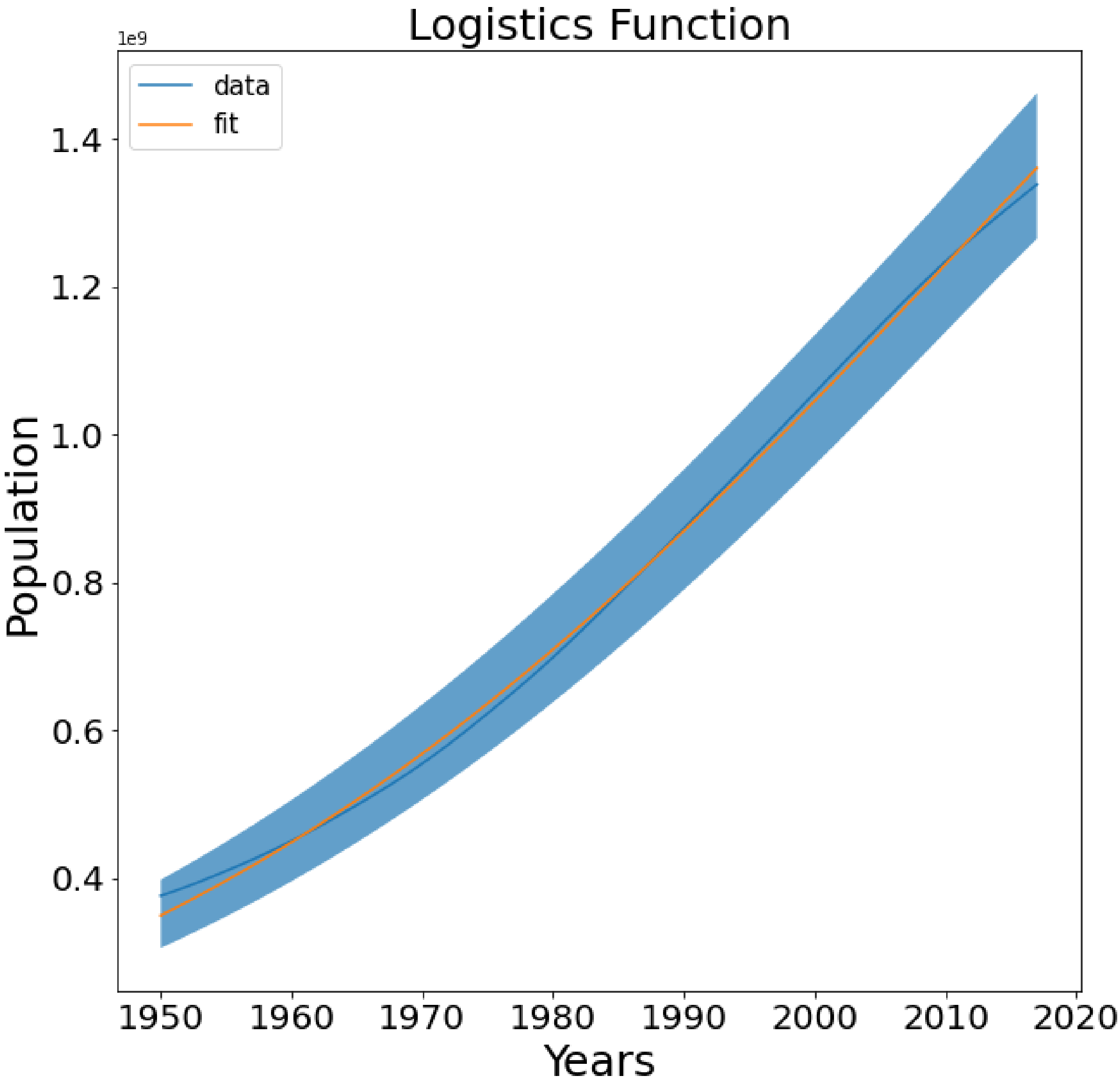
err_ranges()

In [19]:

```
def err_ranges(x, func, param, sigma):
    """
    Calculates the upper and lower limits for the function, parameters and
    sigmas for single value or array x. Functions values are calculated for
    all combinations of +/- sigma and the minimum and maximum is determined.
    Can be used for all number of parameters and sigmas >=1.

    This routine can be used in assignment programs.
    """

    import itertools as iter

    # initiate arrays for lower and upper limits
    lower = func(x, *param)
    upper = lower

    uplow = []   # list to hold upper and lower limits for parameters
    for p,s in zip(param, sigma):
        pmin = p - s
        pmax = p + s
        uplow.append((pmin, pmax))

    pmix = list(iter.product(*uplow))

    for p in pmix:
        y = func(x, *p)
        lower = np.minimum(lower, y)
        upper = np.maximum(upper, y)

    return lower, upper
```

In [20]:
```
# extract the sigmas from the diagonal of the covariance matrix
sigma = np.sqrt(np.diag(c))
print(sigma)

low, up = err_ranges(data["year"], logistics, p, sigma)

plt.figure(figsize=(10,10))
plt.title("Logistics Function", fontsize=25)
plt.plot(data["year"], data["Population"], label="data")
plt.plot(data["year"], data["logistic"], label="fit")

plt.fill_between(data["year"], low, up, alpha=0.7)
plt.legend(loc='upper left',fontsize=15)
plt.xlabel("Years", fontsize=25)
plt.ylabel("Population", fontsize=25)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```

```
[9.44349816e+07 6.01576534e-04 2.44923941e+00]
```



Prediction of the Population

In [21]:
```
# Give Ranges

print("Forcasted Population")
```

```
low, up = err_ranges(2030, logistics, p, sigma)
print("In 2030 between low =", low, "and up =", up)
low, up = err_ranges(2040, logistics, p, sigma)
print("In 2040 between low =", low, "and up =", up)
low, up = err_ranges(2050, logistics, p, sigma)
print("In 2050 between low =", low, "and up =", up)
```

```
Forcasted Population
In 2030 between low = 1489613749.909398 and up = 1707048161.2648458
In 2040 between low = 1650519623.181745 and up = 1877585009.76558
In 2050 between low = 1795631428.9189029 and up = 2027295268.9615705
```

In [ ]: