Analyzing Immigration to Canada from 1980 to 2013

Dataset Source: [International migration flows to and from selected countries - The 2015 revision]().

The dataset contains annual data on the flows of international immigrants as recorded by the countries of destination. The data presents both inflows and outflows according to the place of birth, citizenship or place of previous / next residence both for foreigners and nationals. The current version presents data pertaining to 45 countries.

The Canada Immigration dataset can be fetched from [here]().

```
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Data Collection

```
can_df = pd.read_excel('../data/canada.xlsx',
                       sheet_name='Canada by Citizenship',
                       skiprows=range(20),
                       skipfooter=2
                      )
can_df.head()
```

	Type	Coverage	OdName	AREA	AreaName	REG	RegName	DEV	DevName	1980	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2
0	Immigrants	Foreigners	Afghanistan	935	Asia	5501	Southern Asia	902	Developing regions	16	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2
1	Immigrants	Foreigners	Albania	908	Europe	925	Southern Europe	901	Developed regions	1	...	1450	1223	856	702	560	716	561	539	620	
2	Immigrants	Foreigners	Algeria	903	Africa	912	Northern Africa	902	Developing regions	80	...	3616	3626	4807	3623	4005	5393	4752	4325	3774	4
3	Immigrants	Foreigners	American Samoa	909	Oceania	957	Polynesia	902	Developing regions	0	...	0	0	1	0	0	0	0	0	0	
4	Immigrants	Foreigners	Andorra	908	Europe	925	Southern Europe	901	Developed regions	0	...	0	0	1	1	0	0	0	0	1	

5 rows × 43 columns

```
can_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 43 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Type      195 non-null    object
 1   Coverage  195 non-null    object
 2   OdName    195 non-null    object
 3   AREA      195 non-null    int64
 4   AreaName  195 non-null    object
 5   REG       195 non-null    int64
 6   RegName   195 non-null    object
 7   DEV       195 non-null    int64
 8   DevName   195 non-null    object
 9   1980      195 non-null    int64
 10  1981      195 non-null    int64
 11  1982      195 non-null    int64
 12  1983      195 non-null    int64
 13  1984      195 non-null    int64
 14  1985      195 non-null    int64
 15  1986      195 non-null    int64
 16  1987      195 non-null    int64
 17  1988      195 non-null    int64
 18  1989      195 non-null    int64
 19  1990      195 non-null    int64
 20  1991      195 non-null    int64
 21  1992      195 non-null    int64
 22  1993      195 non-null    int64
 23  1994      195 non-null    int64
 24  1995      195 non-null    int64
```

```
25  1996      195 non-null    int64
26  1997      195 non-null    int64
27  1998      195 non-null    int64
28  1999      195 non-null    int64
29  2000      195 non-null    int64
30  2001      195 non-null    int64
31  2002      195 non-null    int64
32  2003      195 non-null    int64
33  2004      195 non-null    int64
34  2005      195 non-null    int64
35  2006      195 non-null    int64
36  2007      195 non-null    int64
37  2008      195 non-null    int64
38  2009      195 non-null    int64
39  2010      195 non-null    int64
40  2011      195 non-null    int64
41  2012      195 non-null    int64
42  2013      195 non-null    int64
dtypes: int64(37), object(6)
memory usage: 65.6+ KB
```

Data Cleaning

```
can_df.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=True)

can_df.rename(columns = {'OdName':'Country', 'AreaName':'Continent', 'RegName':'Region'},inplace=True)

can_df.head(2)
```

	Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	...	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
0	Afghanistan	Asia	Southern Asia	Developing regions	16	39	39	47	71	340	...	2978	3436	3009	2652	2111	1746	1758	2203	2635	2004
1	Albania	Europe	Southern Europe	Developed regions	1	0	0	0	0	0	...	1450	1223	856	702	560	716	561	539	620	603

2 rows × 38 columns

```
can_df['Total'] = can_df.iloc[:,4:].sum(axis=1)
```

```
can_df.set_index('Country', inplace=True)
```

Column names that are integers (such as the years) might introduce some confusion. For example, when we are referencing the year 2013, one might confuse that when the 2013th positional index. To avoid this ambuigity, let's convert the column names into strings.

```
can_df.columns = list(map(str, can_df.columns))
# to facilitate plotting
years = list(map(str, range(1980, 2014)))
```

I'll change the name of 'United Kingdom of Great Britain and Northern Ireland' countries to 'UK' so it fit the plotting area in a figure

```
new_index = []
for country in can_df.index:
    if country == 'United Kingdom of Great Britain and Northern Ireland':
        country ='UK'
    new_index.append(country)

can_df.index = new_index
```

Exploratory Data Analysis

```
can_df.describe()
```

	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	...	
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	...	195.0
mean	508.394872	566.989744	534.723077	387.435897	376.497436	358.861538	441.271795	691.133333	714.389744	843.241026	...	1320.2
std	1949.588546	2152.643752	1866.997511	1204.333597	1198.246371	1079.309600	1225.576630	2109.205607	2443.606788	2555.048874	...	4425.9
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.500000	0.500000	1.000000	1.000000	...	28.5
50%	13.000000	10.000000	11.000000	12.000000	13.000000	17.000000	18.000000	26.000000	34.000000	44.000000	...	210.0
75%	251.500000	295.500000	275.000000	173.000000	181.000000	197.000000	254.000000	434.000000	409.000000	508.500000	...	832.0
max	22045.000000	24796.000000	20620.000000	10015.000000	10170.000000	9564.000000	9470.000000	21337.000000	27359.000000	23795.000000	...	42584.0

8 rows × 35 columns

Making function to avoid repeating the code

```
def make_basic_continent_stat(continent):
    """a function that takes a name of a continent and return the basic statistics of the immigration numbers
        of the continent"""
    print(f'{continent} Total Immigration Basic Staistics')
    df = can_df[can_df['Continent']==continent]['Total'].describe()
    return df

def make_basic_stat(continent,k):
    """a function that takes a name of a continent and return the basic statistics of the immigration numbers
        of the top 3 countries in that continent"""
    df = (can_df[can_df['Continent']==continent].
         sort_values('Total',ascending=False).
         head(k).
         loc[:,years].T).describe()
    return df

def make_bar_pie_df(continent,k):
    """a function that takes a name of a continent and return a dataframe that shows the total number
        of immigrants of the top 3 countries in a continent with those countries as index. This dataframe
        is suaitbale for bar chart and pie chart plots"""
    df = can_df[can_df['Continent']==continent]['Total'].\
         to_frame().\
         sort_values('Total',ascending=False).\
         head(k)
    return df
```

```
def make_line_box_df(continent,k):
    """a function that takes a name of a continent and return a dataframe that show the name of the
        country and the number of immigrants for the top 3 countries in the continent with years as index.
        This dataframe is suitable for box and line plots"""
    df = (can_df[can_df['Continent']==continent].
         sort_values('Total',ascending=False).
         head(k).
         loc[:,years].T)

    df.columns.name=''

    df.reset_index(inplace=True)
    df.rename(columns={'index':'Year'}, inplace=True)

    df= df.melt(id_vars='Year',var_name='Country',value_name='Number of Immigrants').\
          set_index('Year')
    df.index = [int(val) for val in df.index]

    return df
```

```
def make_vis(df1,df2,continent,k):
    """a function takes in 2 dataframes and returns visualizations of the immigration numbers in df1 and df2"""
    chart_order = df1.index
    plt.figure(figsize=(26,18), facecolor='#C4A484')
    plt.suptitle(f"Visalization of Immigrants Numbers from Top {k} Countries in {continent} (1980-2013)",size=22)

    plt.subplot(2,2,1)
    sns.barplot(x=df1.index,
                y='Total',
                data=df1,
                order=chart_order)
    plt.title('Bar Plot', size=20)
    plt.xlabel('Country',size=18,color='black')
    plt.ylabel('Number of Immigrants',size=18,color='black')
    plt.xticks(size = 14)
    plt.yticks(size = 14)


    plt.subplot(2,2,2)
    plt.pie(x = df1['Total'],
            labels = df1.index,
            radius= 1.1,
            startangle= 90,
            counterclock= False,
            autopct= '%2.1F%%',
            textprops={'fontsize': 18})
    plt.title('Pie Plot', size=20)
```

```
    plt.subplot(2,2,3)
    sns.boxplot(x='Number of Immigrants',
                y = 'Country',
                data=df2)
    plt.title('Box Plot', size = 20)
    plt.xlabel('Number of Immigrants', size = 18,color='black')
    plt.ylabel('Country', size = 18,color='black')
    plt.xticks(size = 14)
    plt.yticks(size = 14)


    plt.subplot(2,2,4)
    ax = sns.lineplot(data=df2,
                      x = df2.index,
                      y='Number of Immigrants',
                      hue='Country')
    plt.title('Line Plot',size=20)
    plt.xlabel('Year', size=18,color='black')
    plt.ylabel('Number of Immigrants',size=18,color='black')
    plt.xticks(size = 14)
    plt.yticks(size = 14)
    plt.setp(ax.get_legend().get_texts(), fontsize='18')
    plt.setp(ax.get_legend().get_title(), fontsize='22')
```

Analysis of Immigration from Africa

Continent Basic Statistics

```
make_basic_continent_stat('Africa')
```

```
Africa Total Immigration Basic Sttaistics

count       54.000000
mean     11462.000000
std      18410.630212
min          2.000000
25%        690.250000
50%       2805.500000
75%      14891.500000
max      72745.000000
Name: Total, dtype: float64
```

Top 3 Countries in Africa Basic Statistics

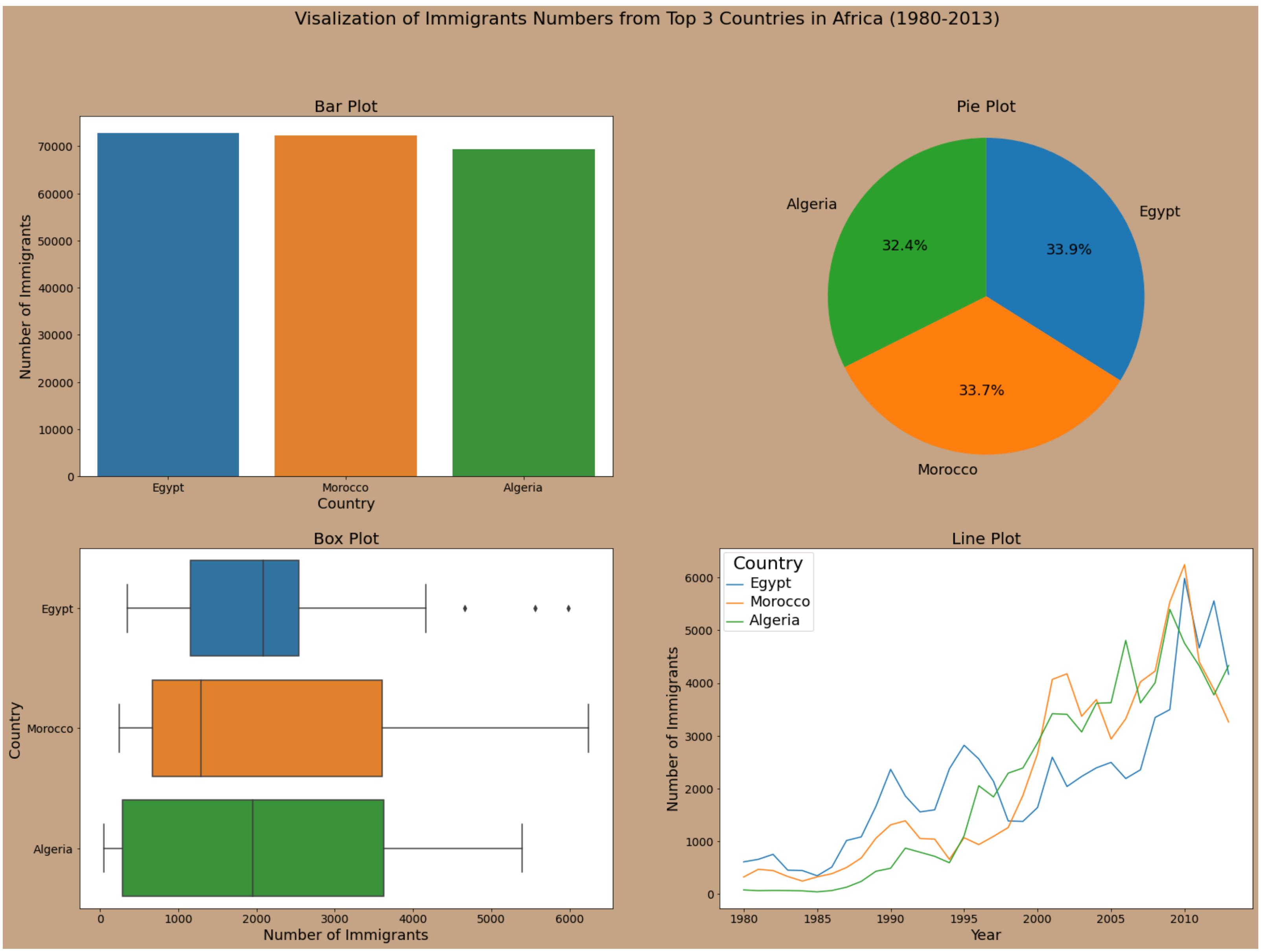
```
make_basic_stat('Africa',3)
```

Country	Egypt	Morocco	Algeria
count	34.000000	34.000000	34.000000
mean	2139.558824	2125.500000	2042.323529
std	1393.809478	1724.108171	1771.709374
min	348.000000	248.000000	44.000000
25%	1158.250000	666.250000	290.000000
50%	2088.000000	1287.500000	1948.000000
75%	2544.750000	3606.750000	3621.250000
max	5982.000000	6242.000000	5393.000000

Visualization of the Immigration Statistics of Top 3 Countries in Africa

Visualization of the Immigration Statistics of Top 3 Countries in Africa

```
africa_bar_pie_df = make_bar_pie_df('Africa',3)
africa_line_box_df = make_line_box_df('Africa',3)
make_vis(africa_bar_pie_df,africa_line_box_df,'Africa',3)
```



Anslysis of Immigration from Latin America and the Caribbean

Latin America and the Caribbean Basic Statistics

```
make_basic_continent_stat('Latin America and the Caribbean')
```

```
Latin America and the Caribbean Total Immigration Basic Sttaistics

count        33.000000
mean      23186.303030
std       28238.853615
min         653.000000
25%        3205.000000
50%       11193.000000
75%       29659.000000
max      106431.000000
Name: Total, dtype: float64
```

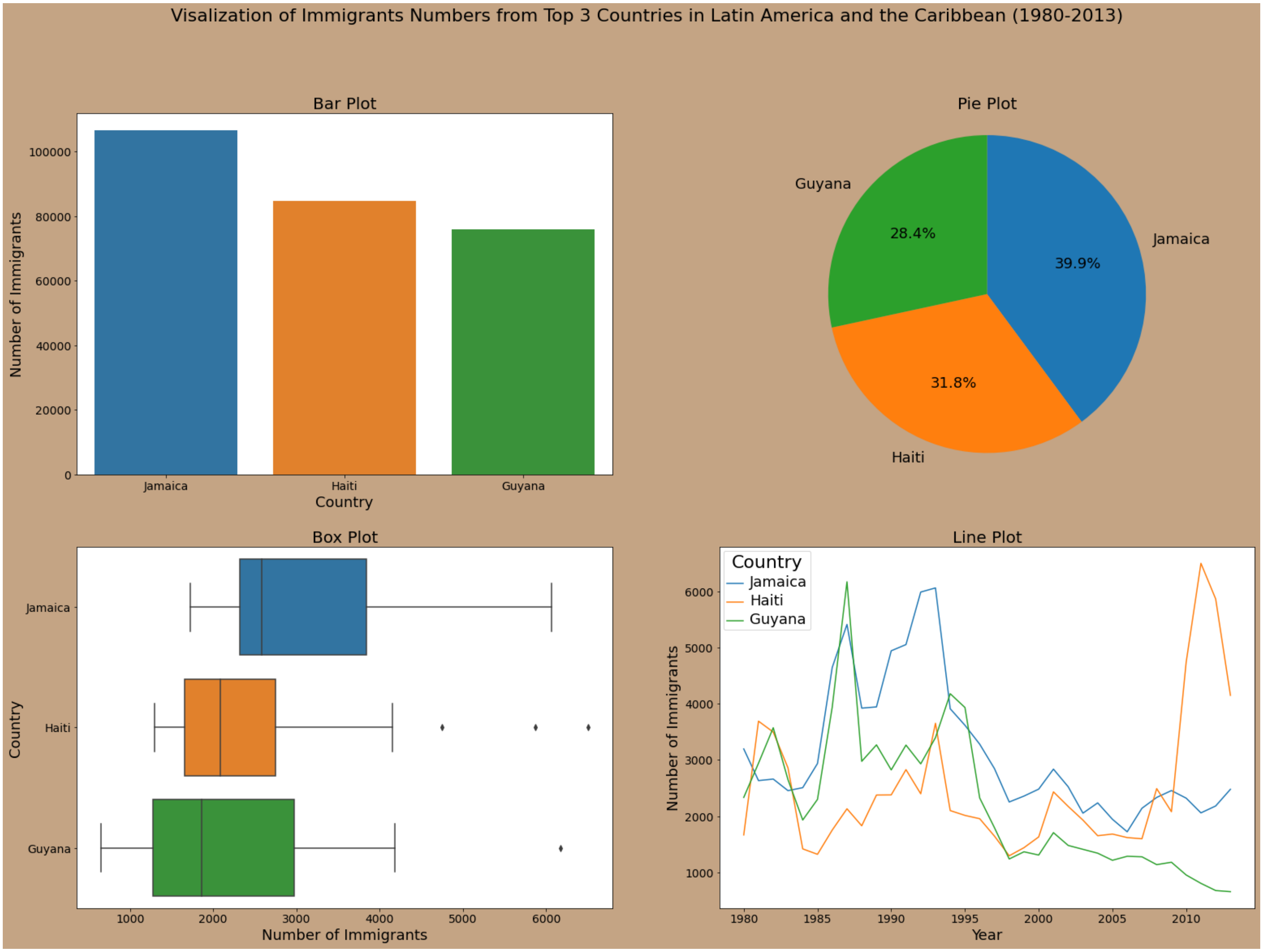
Top 3 Countries in Latin America and the Caribbean Basic Statistics

```
make_basic_stat('Latin America and the Caribbean',3)
```

Country	Jamaica	Haiti	Guyana
count	34.000000	34.000000	34.000000
mean	3130.323529	2494.500000	2228.970588
std	1202.308001	1254.169247	1249.060531
min	1722.000000	1295.000000	656.000000
25%	2324.250000	1655.500000	1279.250000
50%	2579.000000	2090.000000	1863.500000
75%	3839.500000	2744.500000	2968.500000
max	6065.000000	6503.000000	6174.000000

Visualization of the Immigration Statistics of Top 3 Countries in Latin America and the Caribbea

```
latin_america_bar_pie_df = make_bar_pie_df('Latin America and the Caribbean',3)
latin_america_line_box_df = make_line_box_df('Latin America and the Caribbean',3)
make_vis(latin_america_bar_pie_df,latin_america_line_box_df,'Latin America and the Caribbean',3)
```



Anslysis of Immigration from Europe ¶

Europe Basic Statistics

```
make_basic_continent_stat('Europe')
```

```
Europe Total Immigration Basic Sttaistics

count        43.000000
mean      32812.720930
std       87055.294354
min           5.000000
25%        2132.500000
50%        5963.000000
75%       22239.500000
max      551500.000000
Name: Total, dtype: float64
```

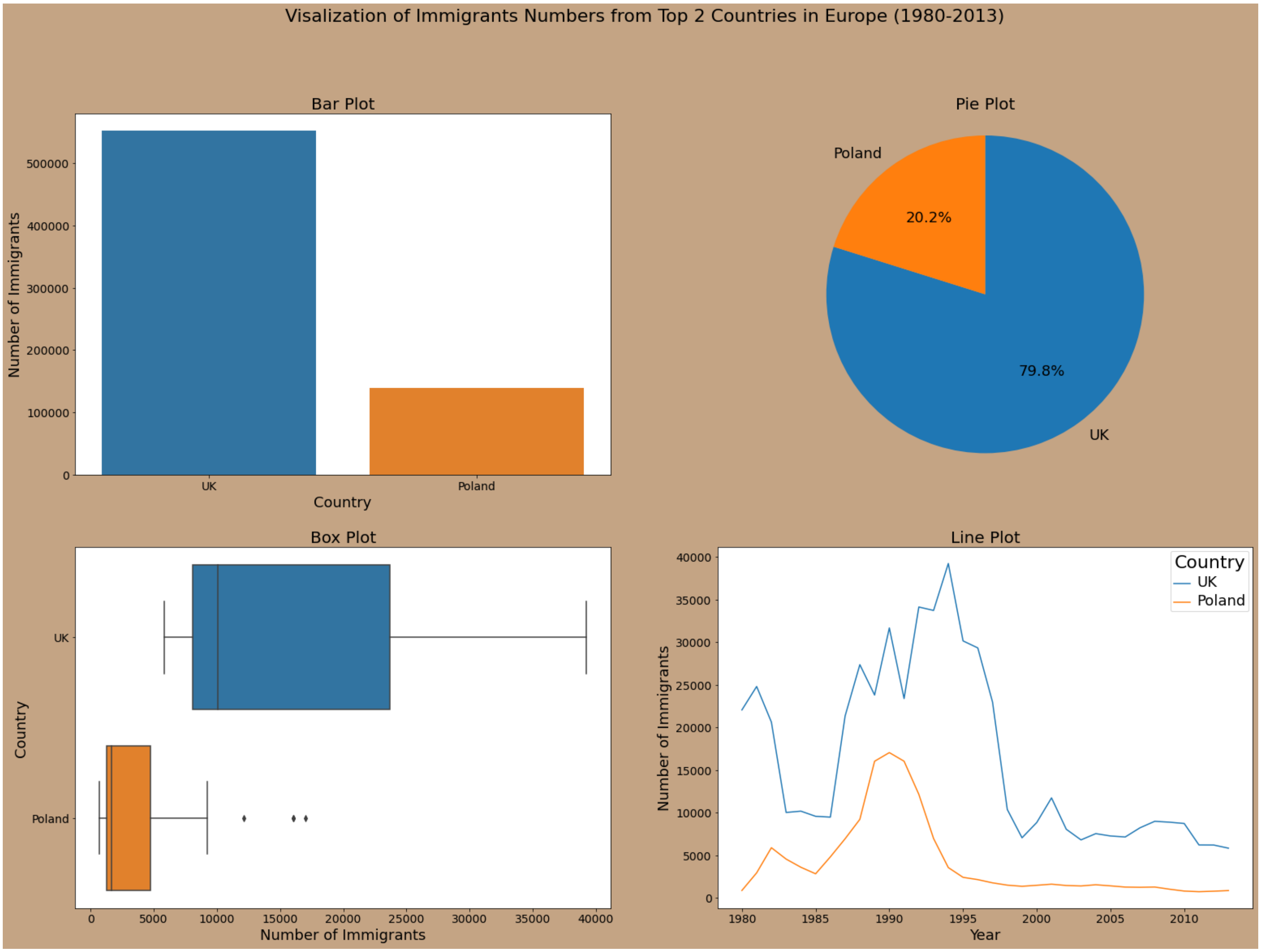
Top 2 Countries in Europe Basic Statistics

```
make_basic_stat('Europe',2)
```

	UK	Poland
count	34.000000	34.000000
mean	16220.588235	4095.323529
std	10267.724908	4679.340654
min	5827.000000	720.000000
25%	8088.500000	1288.750000
50%	10092.500000	1679.500000
75%	23691.250000	4742.500000
max	39231.000000	17040.000000

Visualization of the Immigration Statistics of Top 2 Countries in Europe

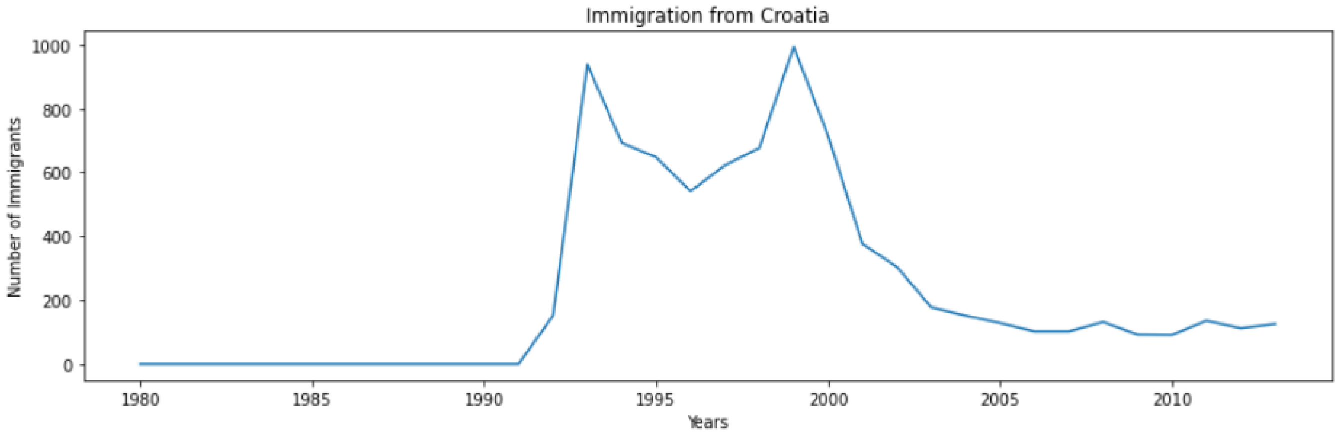
```
europe_bar_pie_df = make_bar_pie_df('Europe',2)
europe_line_box_df = make_line_box_df('Europe',2)
make_vis(europe_bar_pie_df,europe_line_box_df,'Europe',2)
```



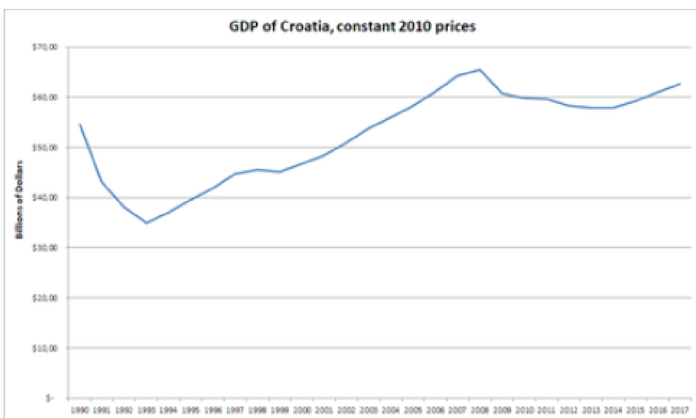
Anslysis of Immigration from Balkan Countries

Exploring Immigration from Croatia

```
can_df.loc['Croatia', years].plot(figsize=(14,4), title='Immigration from Croatia')
plt.xlabel('Years')
plt.ylabel('Number of Immigrants')
plt.show()
```



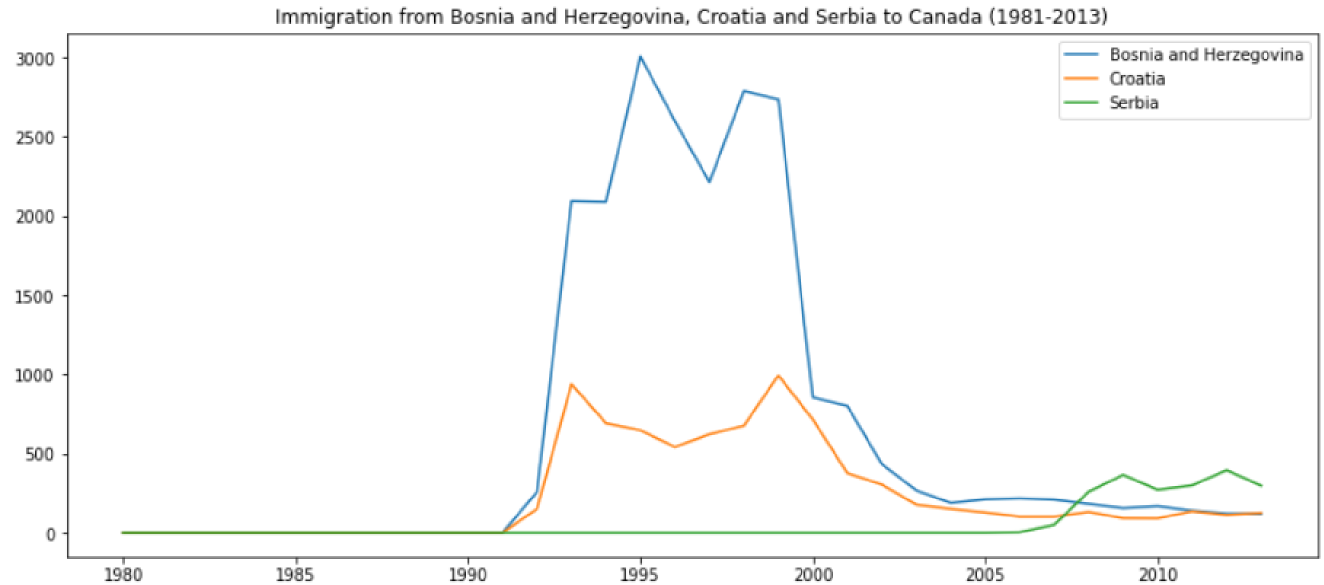

```
from IPython import display
display.Image("https://upload.wikimedia.org/wikipedia/commons/thumb/f/f3/GDP_of_Croatia_at_constant_prices.png/440px-GDP_of_Croat
```



It can be seen that immigration from Croatia began in the early 1990s (during the war in Yugoslavia), then number of immigrants started to decrease sharply after the war. Since around 1999, Croatia's GDP started to incease steadily. Perhaps that was why the number of immigrants from Croatia statred to decrease drastically. Interestingly, the immigration numbers were not affected much by the recession of 2008.

Comparing the Immigration numbers of the 3 major countries in Balkan War:
There were 3 major countries involved in that war; Bosnia and Herzegovina, Croatia and Serbia

```
balkan_df = can_df.loc[['Bosnia and Herzegovina','Croatia','Serbia'],years].T
balkan_df.plot(figsize=(14,6), title='Immigration from Bosnia and Herzegovina, Croatia and Serbia to Canada (1981-2013)')
plt.show()
```



There was no immigration from the 3 countries before 1991 because there were no countries with those names and the Balkan war was the Independence war for Bosnia and Herzegovina and Croatia.

Bosnia and Herzegovina has much higher immgration numbers than Croatia and Serbia in the period of early 1990s till the second half of 2000s.

Immigration from Bosnia and Herzegovina and Croatia started to drop significantly in the second half of 1990s and the trend continued through the 2000s although with much lower slope.

Serbia immgration started to take off in the second half of the 2000s with a bit of fluctuation. In the last few years, Serbia immigration exceeded those of both Bosnia and Herzegovina and Croatia

Analysis of the Immigration from North America

First, let check what are the countries listed under Northern America

```
can_df.Continent.value_counts().index
```

```
Index(['Africa', 'Asia', 'Europe', 'Latin America and the Caribbean',
       'Oceania', 'Northern America'],
      dtype='object')
```

```
can_df[can_df.Continent == 'Northern America'].index
```

```
Index(['Canada', 'United States of America'], dtype='object', name='Country')
```

Interestingly, Canada has a row in the data although the data is for the immigration numbers to Canada. Moreover, according to the data source, Mexico is not among North America countries...what?!! Continent is defined by geography not language.Let's fix that.

```
can_df.drop('Canada', axis = 0, inplace=True)
```

```
can_df.loc['Mexico', 'Continent'] = 'Northern America'
```

```
can_df[can_df['Continent']=='Northern America']
```

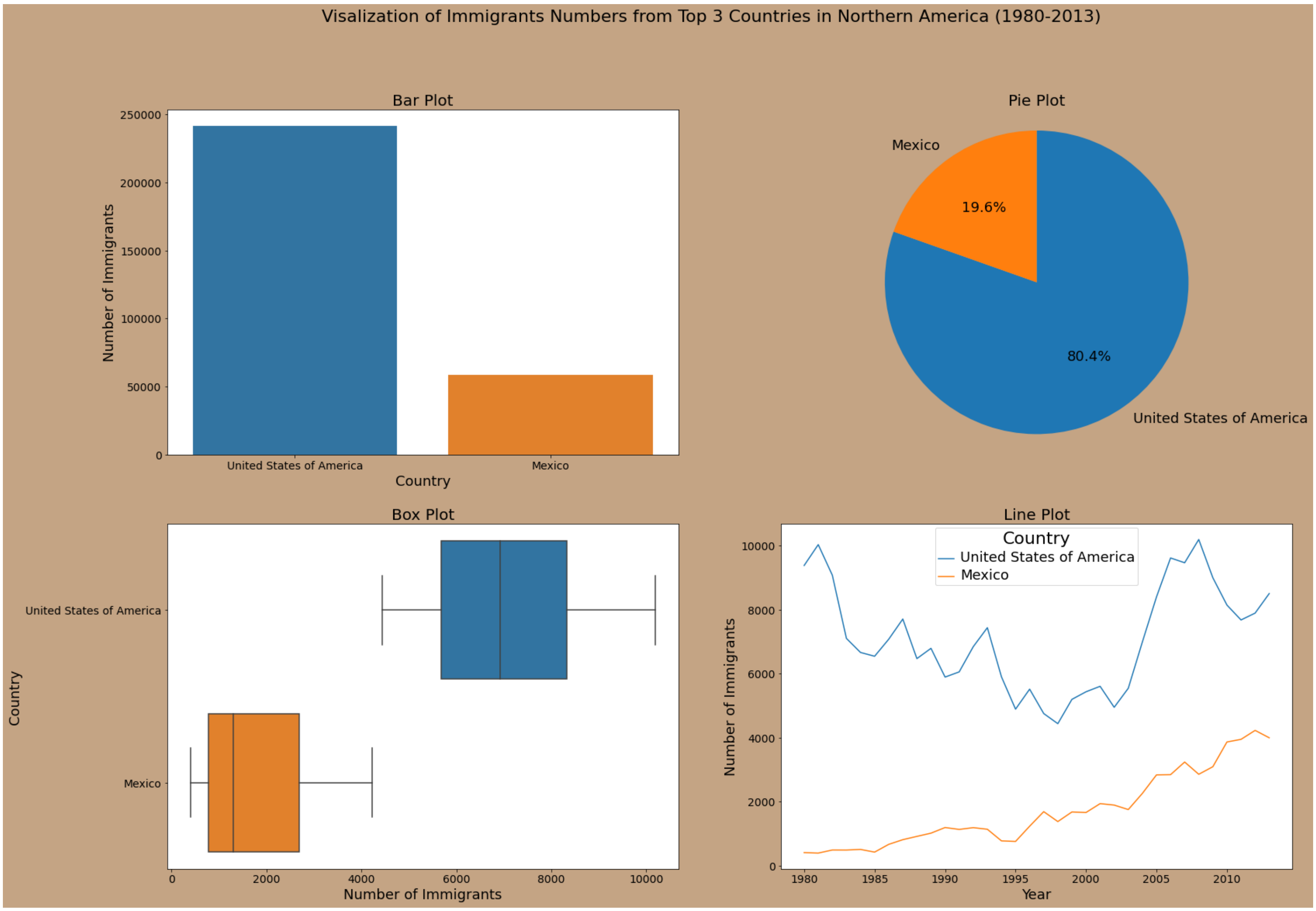
Country	Continent	Region	DevName	1980	1981	1982	1983	1984	1985	1986	...	2005	2006	2007	2008	2009	2010	2011	2012	2013	Total
Mexico	Northern America	Central America	Developing regions	409	394	491	490	509	425	667	...	2837	2844	3239	2856	3092	3865	3947	4227	3996	58712
United States of America	Northern America	Northern America	Developed regions	9378	10030	9074	7100	6661	6543	7074	...	8394	9613	9463	10190	8995	8142	7676	7891	8501	241122

2 rows × 38 columns

Cool! Now, let's explore the relevant data.

Visualization of the Immigration Statistics from Northern America

```
make_vis(make_bar_pie_df('Northern America'),make_line_box_df('Northern America'),'Northern America')
```

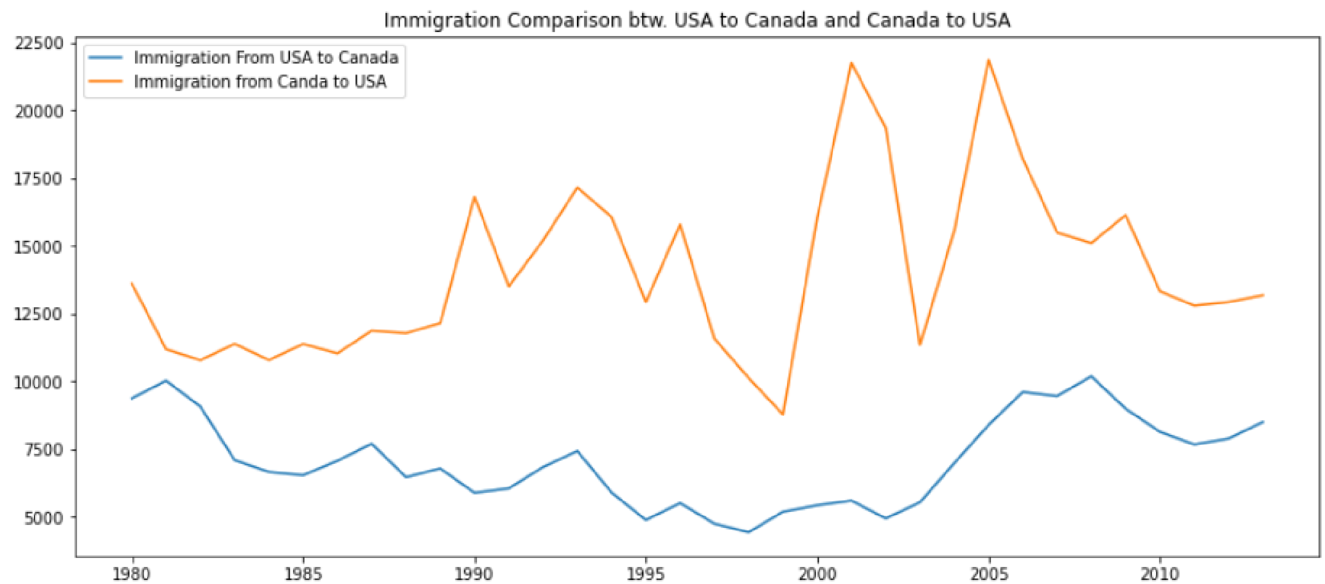


That makes sense. The Untied States of America is much closer to Canda than Mexico. Moreover, USA's citizens can enter Canada and live there until there are qualified for citizenship without any problem due to the agreements bteween the two countries' governments. Furthermore, the culture of Canada and language is close to those of the United States of America.

You know what would be more interesting? Yeah... to compare the number of immigrants from Canada to USA and compare it to that from USA to Canada. That is why I downloaded USA Immigration data.

```python
usa_df = pd.read_excel('../data/United States of America.xlsx',
                       sheet_name='USA by Place of birth',
                       skiprows=range(20),
                       skipfooter=2
                      )
usa_df.drop(['AREA','REG','DEV','Type','Coverage'], axis=1, inplace=True)

usa_df.rename(columns = {'OdName':'Country', 'AreaName':'Continent', 'RegName':'Region'},inplace=True)

usa_df['Total'] = can_df.iloc[:,4:].sum(axis=1)

usa_df.set_index('Country', inplace=True)

usa_df.columns = list(map(str, can_df.columns))
```

```python
can_df.loc['United States of America', years].plot(label='Immigration From USA to Canada')
usa_df.loc['Canada', years].plot(label = 'Immigration from Canda to USA')
plt.title('Immigration Comparison btw. USA to Canada and Canada to USA')
plt.legend()
plt.show()
```



That makes sense. USA has bigger economy and warmer climate.

Analysis of the Immigration from Asia

Asia Basic Statistics

```python
make_basic_continent_stat('Asia')
```

```
Asia Total Immigration Basic Sttaistics

count        49.000000
mean      67710.081633
std      153311.733183
min          30.000000
25%        1126.000000
50%        8490.000000
75%       58639.000000
max      691904.000000
Name: Total, dtype: float64
```

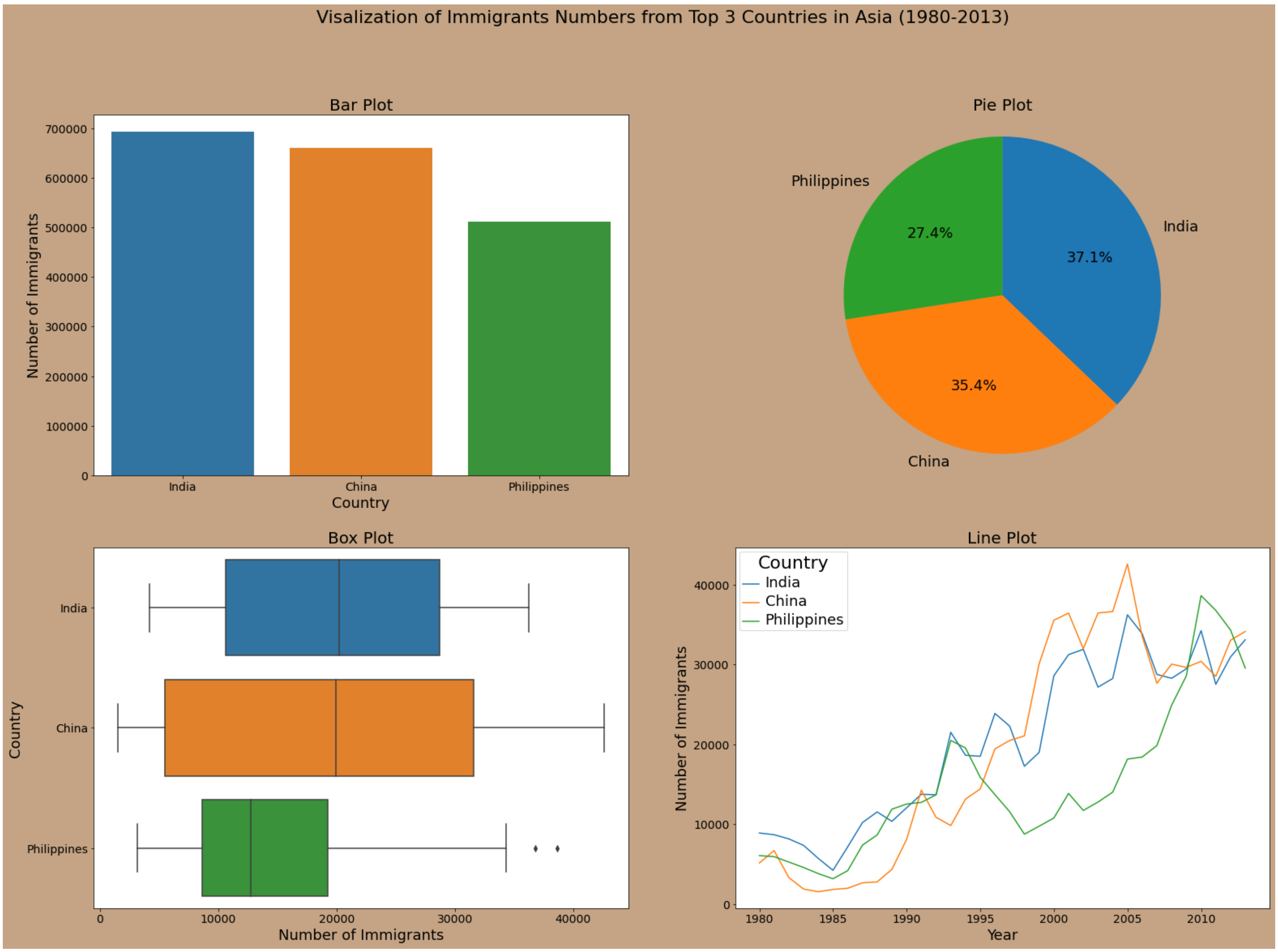
Basic Statistics of Top 3 Countries in Asia

```
make_basic_stat('Asia',3)
```

	India	China	Philippines
count	34.000000	34.000000	34.000000
mean	20350.117647	19410.647059	15040.911765
std	10007.342579	13568.230790	9506.754936
min	4211.000000	1527.000000	3150.000000
25%	10637.750000	5512.750000	8663.000000
50%	20235.000000	19945.000000	12738.000000
75%	28699.500000	31568.500000	19249.000000
max	36210.000000	42584.000000	38617.000000

Visualization of the Immigration Statistics of Top 3 Countries in Asia

```
asia_bar_pie_df = make_bar_pie_df('Asia',3)
asia_line_box_df = make_line_box_df('Asia',3)
make_vis(asia_bar_pie_df,asia_line_box_df,'Asia',3)
```



Time Series Analysis for the Immigration from Biggest Immigrants Country

```
import datetime as dt
import itertools
import statsmodels.graphics.tsaplots as sgt
import statsmodels.tsa.stattools as sts
from statsmodels.tsa.ar_model import AR
from statsmodels.tsa.arima_model import ARIMA
from pmdarima.arima.utils import ndiffs
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_squared_error
```

Identifying the Biggest Country Immigration COuntry

```
can_df['Total'].sort_values(ascending=False).head(1).index
```

```
Index(['India'], dtype='object', name='Country')
```

```
ts = can_df.loc['India',years].to_frame()

ts.head()
```

	India
1980	8880
1981	8670
1982	8147

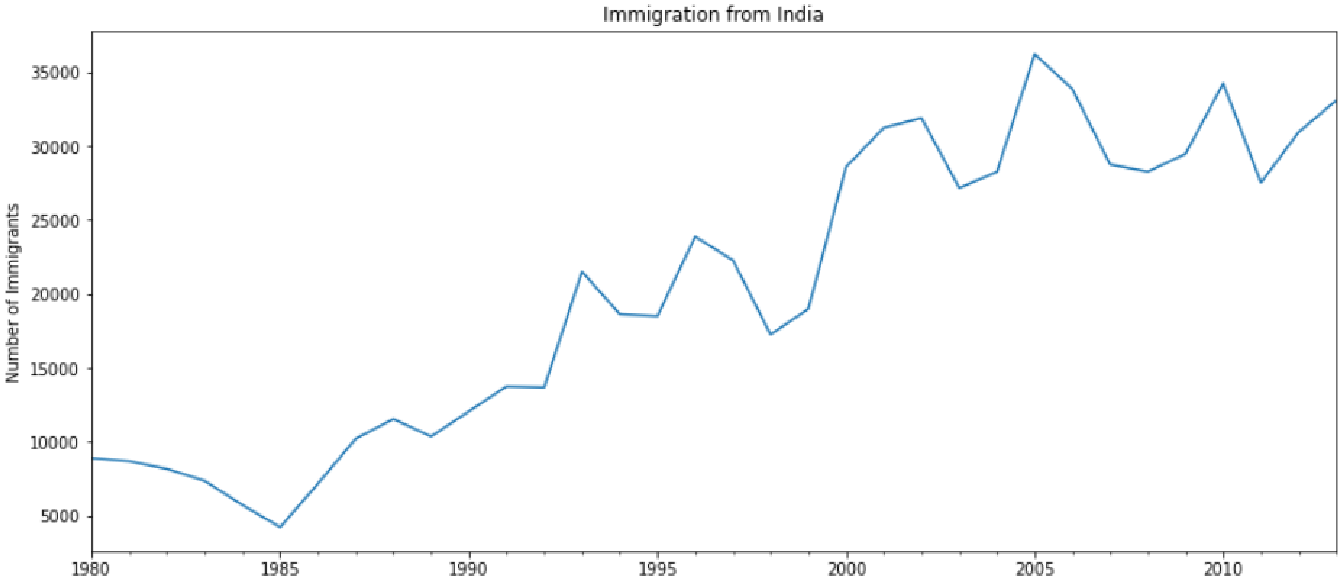
Converting the index to datetime type

```
# ts.index = [dt.datetime.strptime(val,"%Y") for val in ts.index]
ts.index = [dt.datetime(int(val),12,31) for val in ts.index]
# ts.index = ts.index.to_period('Y')
ts.rename(columns = {'India':'Number of Immigrants'}, inplace=True)
ts.head()
```

	Number of Immigrants
1980-12-31	8880
1981-12-31	8670
1982-12-31	8147
1983-12-31	7338
1984-12-31	5704

Plotting the Time Series

```
ts['Number of Immigrants'].plot(figsize=(14,6),title='Immigration from India')
plt.xlabel('Years')
plt.ylabel('Number of Immigrants')
plt.show()
```



Checking the Stationarity of the Time Series

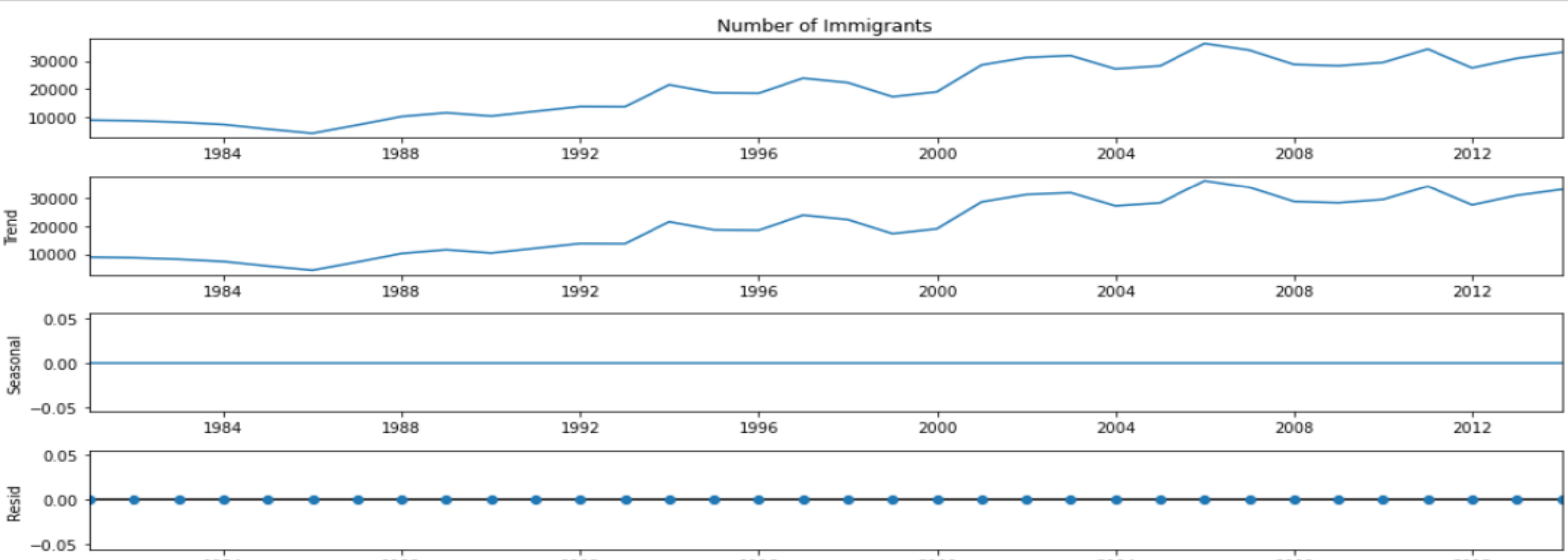
```
# it can be seen from the previous plot that the mean is not constant. So, this time series is not
# stationary
t_stat,p_value,lags, observation_num,t_critical_vals,ic = sts.adfuller(ts['Number of Immigrants'])
print('P_value: ',p_value)
if p_value <= 0.05:
    print('There is a significant evidence that this time series is stationary')
else:
    print('There is no significant evidence that this time series is stationary')
```

```
P_value:  0.8392746310524928
There is no significant evidence that this time series isstationary
```

Since the time series is not stationary, then AR, MA or ARMA are not suitable models to use here.I'll difference the time series to make it stationary.

Checking the Seasonality of the Time Series

```
plt.rcParams["figure.figsize"] = (14,6)
seasonal_decompose(ts['Number of Immigrants']).plot()
plt.show()
plt.close()
```

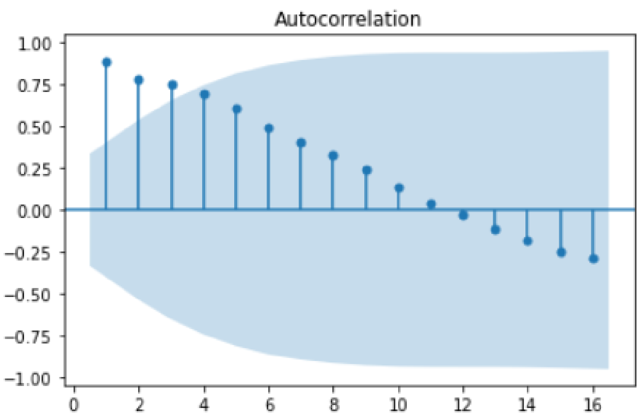


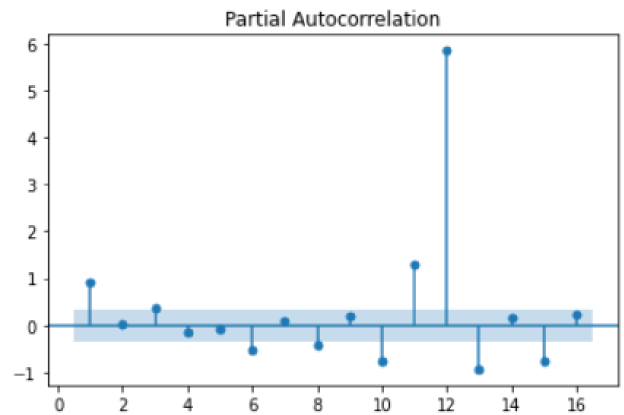
It can be seen that there is not seasonality in this time series

Plotting the Autocorrelation Function and the Partial Autocorrelation Function

```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))
# ax1.plot(ts['Number of Immigrants'])
sgt.plot_acf(ts['Number of Immigrants'],zero=False, ax=ax1)
sgt.plot_pacf(ts['Number of Immigrants'],zero=False, ax=ax2)
plt.show()
plt.close()
```

```
C:\ProgramData\Miniconda3\lib\site-packages\statsmodels\regression\linear_model.py:1434: RuntimeWarning: invalid value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```





Determining p, d, q values

Determining d

```
diff_num = ndiffs(ts['Number of Immigrants'], test='adf')
diff_num
```

```
1
```

```
# it can be seen taht d = 1
```

Converting the time series to stationary

```
ts_diff = ts.diff(1)[1:]
ts_diff.rename(columns = {'Number of Immigrants':' Difference in NUmber of Immigrants'}, inplace=True)
ts_diff.head()
```

	Difference in NUmber of Immigrants
1981-12-31	-210
1982-12-31	-523
1983-12-31	-809
1984-12-31	-1634
1985-12-31	-1493

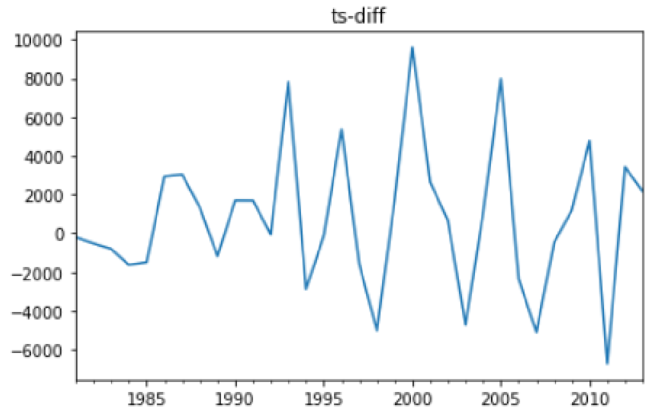
Checking the Stationarity of the new Time Series

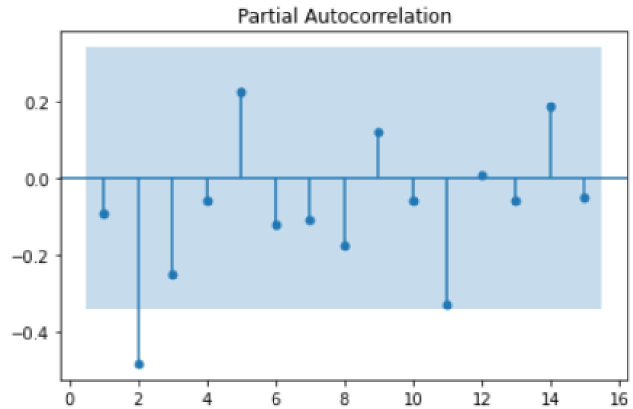
```
t_stat,p_value,lags, observation_num,t_critical_vals,ic = sts.adfuller(ts_diff)
print('P_value: ',p_value)
if p_value <= 0.05:
    print('There is a significant evidence that this time series is stationary')
else:
    print('There is no significant evidence that this time series isstationary')
```

```
P_value:  1.873804434239198e-08
There is a significant evidence that this time series is stationary
```

Determining p

```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))

ts_diff.plot(ax = ax1, title = 'ts-diff', legend = False)
sgt.plot_pacf(ts_diff,zero=False, lags=15, ax = ax2)
plt.show()
plt.close()
```

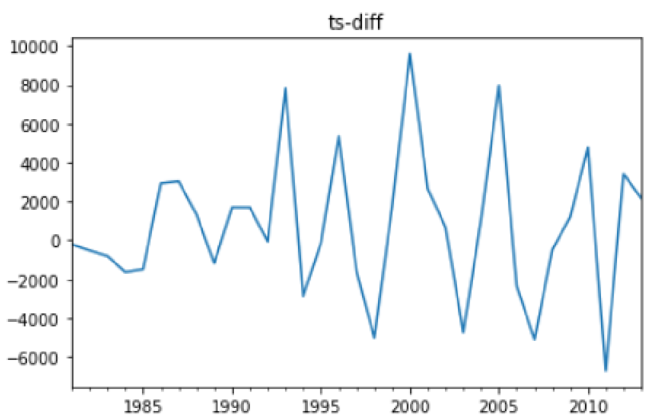


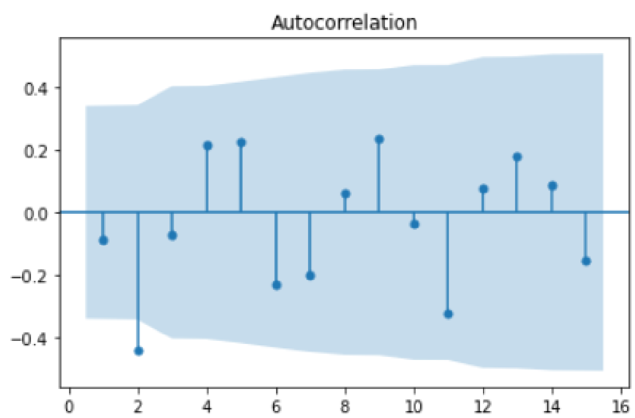



```
# from the partial autocorrelation function, it can be seen that p = 2
```

Determining q

```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))

ts_diff.plot(ax = ax1, title = 'ts-diff', legend = False)
sgt.plot_acf(ts_diff,zero=False, lags=15, ax = ax2)
plt.show()
plt.close()
```






```
# it can be seen from the partial autocorrelation plot that q = 2
```

Splitting the Data

```
x_train = ts['Number of Immigrants'].values[0:30] # 30 data point
x_test = ts['Number of Immigrants'].values[30:] # 3 data point
```

Fitting the ARIMA Model

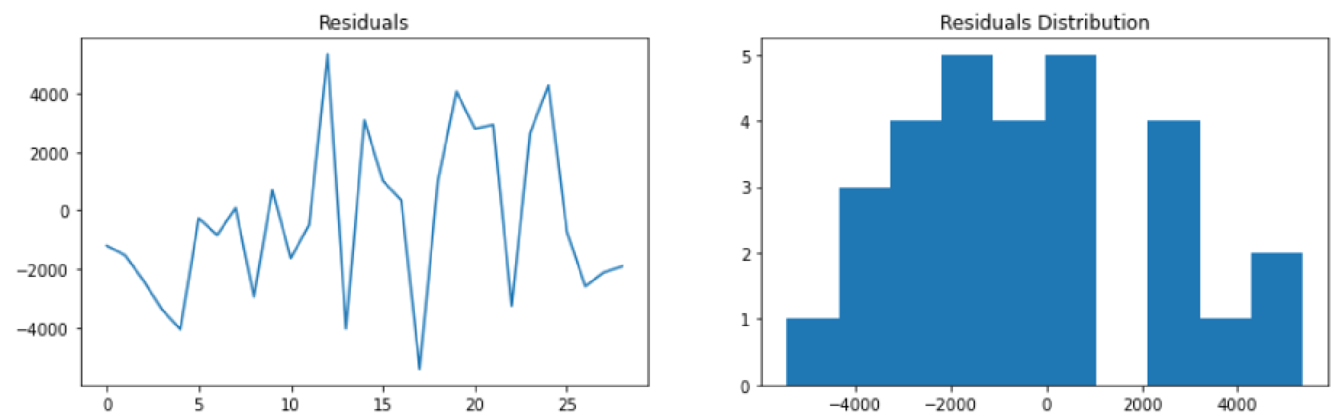
```
model = ARIMA(x_train, order = (2,1,2))
model_fit = model.fit(disp=0)
print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                   29
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -271.500
Method:                       css-mle   S.D. of innovations           2571.618
Date:                Fri, 22 Apr 2022   AIC                            555.000
Time:                        10:36:09   BIC                            563.204
Sample:                             1   HQIC                           557.569

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        991.6300    108.981      9.099      0.000     778.030    1205.230
ar.L1.D.y     -0.0266      0.196     -0.136      0.892      -0.410       0.357
ar.L2.D.y      0.0566      0.201      0.281      0.779      -0.338       0.451
ma.L1.D.y   3.899e-07      0.153   2.55e-06      1.000      -0.300       0.300
ma.L2.D.y     -1.0000      0.153     -6.533      0.000      -1.300      -0.700
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -3.9760           +0.0000j            3.9760            0.5000
AR.2            4.4464           +0.0000j            4.4464            0.0000
MA.1           -1.0000           +0.0000j            1.0000            0.5000
MA.2            1.0000           +0.0000j            1.0000            0.0000
```

Plotting the Residual Errors

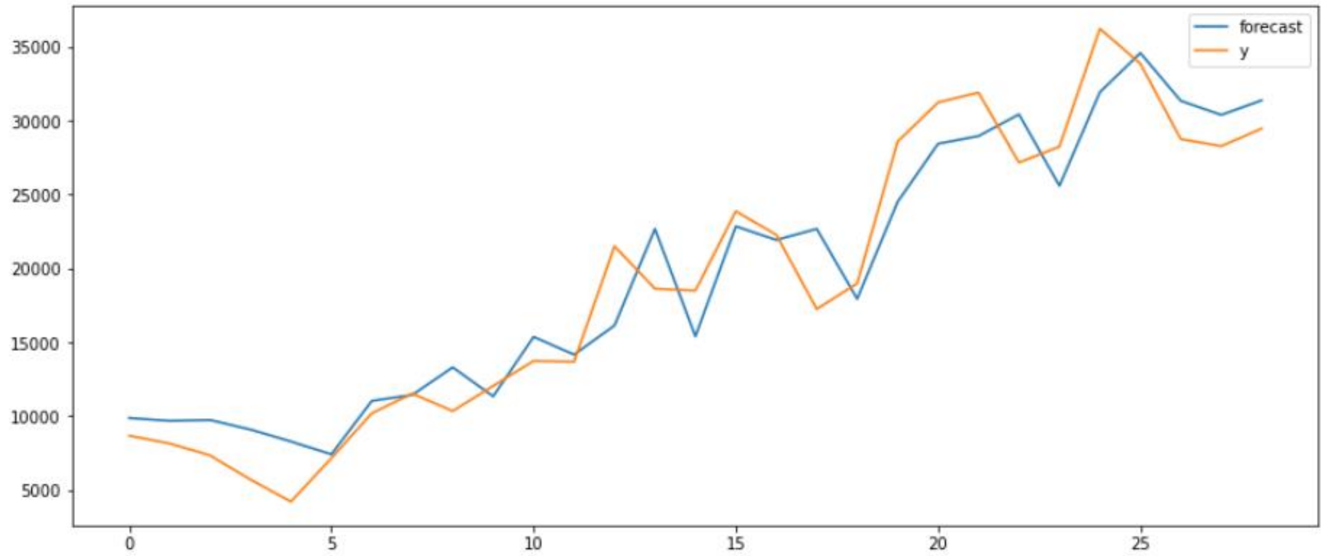
```
residuals = pd.DataFrame(model_fit.resid)
```

```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(14,4))

ax1.plot(residuals)
ax1.set_title('Residuals')
ax2.hist(residuals, bins = 10)
ax2.set_title('Residuals Distribution')
plt.show()
```



Plotting the Model Predicted Values with the Actual Values

```
model_fit.plot_predict();
```



Forecasting

```
steps = 4
fc, se, conf = model_fit.forecast(steps)
```

```
fc
```

```
array([32476.05076172, 35335.85099    , 36392.50618337, 37488.07867502])
```

```
conf
```

```
array([[27435.77290726, 37516.32861619],
       [28302.0196722 , 42369.68230781],
       [29356.97708134, 43428.0352854 ],
       [30447.23913786, 44528.91821217]])
```

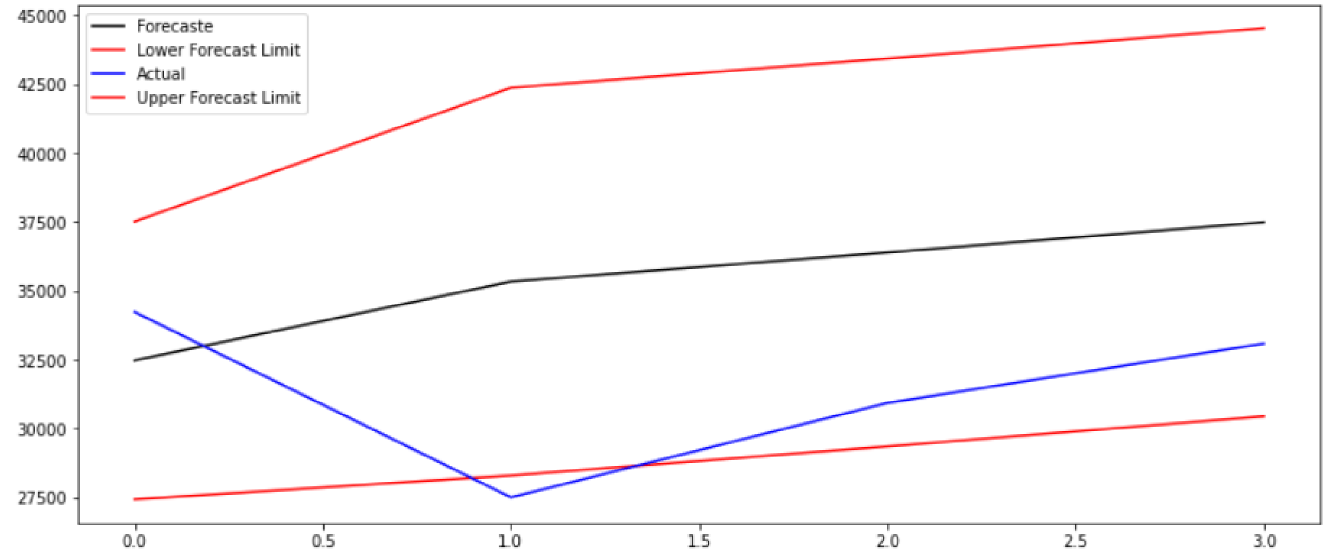
```
conf[:,0]
```

```
array([27435.77290726, 28302.0196722 , 29356.97708134, 30447.23913786])
```

```
result =  pd.DataFrame({ 'Forecaste':fc,
                        'Lower Forecast Limit':conf[:,0],
                        'Actual':x_test,
                        'Upper Forecast Limit':conf[:,1]})
result
```

	Forecaste	Lower Forecast Limit	Actual	Upper Forecast Limit
0	32476.050762	27435.772907	34235	37516.328616
1	35335.850990	28302.019672	27509	42369.682308
2	36392.506183	29356.977081	30933	43428.035285
3	37488.078675	30447.239138	33087	44528.918212

```
result.plot(color=['black','red','blue','red'])
plt.show()
```



This model is not very accutate since it goes out of the confidence interval.

Alternative Better Model using Grid Search

```
p=d=q=range(0,3)
pdq = list(itertools.product(p,d,q))
```

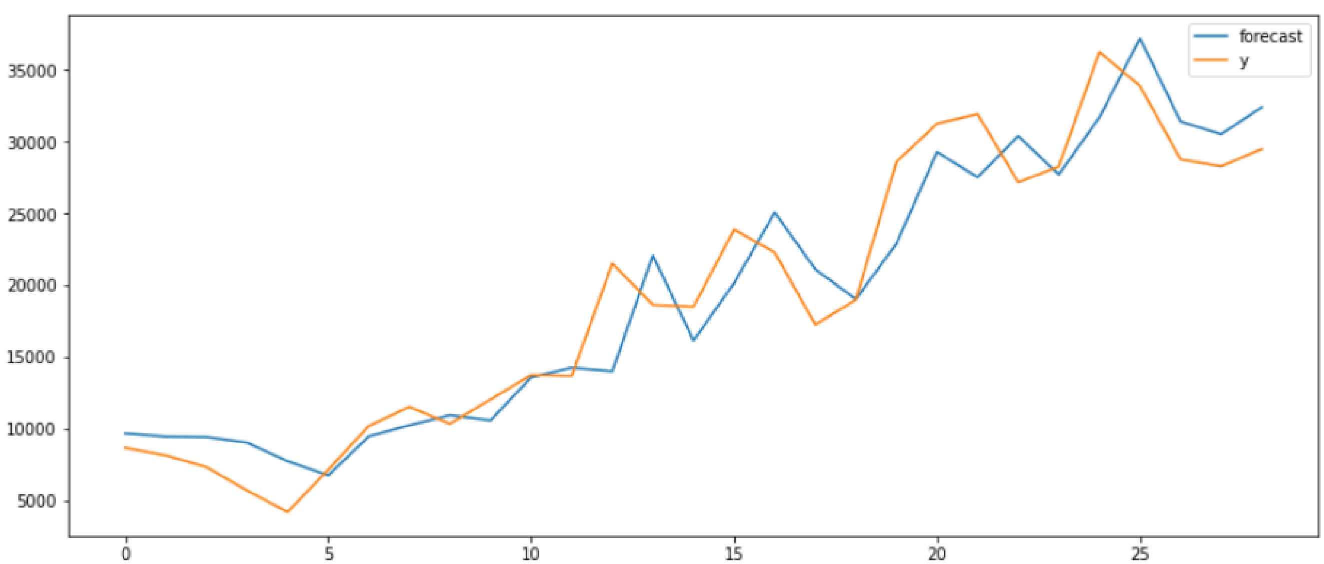
```
param_aic = {}
for param in pdq:
    try:
        model_arima = ARIMA(x_train, order = param)
        model_arima_fit = model_arima.fit()
        param_aic[(param[0],1,param[2])] = model_arima_fit.aic
    except:
        continue
```

```
best_param = min(param_aic, key= param_aic.get)
best_param,param_aic[best_param]
```

```
((2, 1, 1), 544.1844911434939)
```

Plotting the Mdodel Predicted Values with the Actual Values

```
model_fit1.plot_predict();
```



```
model1 = ARIMA(x_train, order = best_param)
model_fit1 = model1.fit()
print(model_fit1.summary())
```

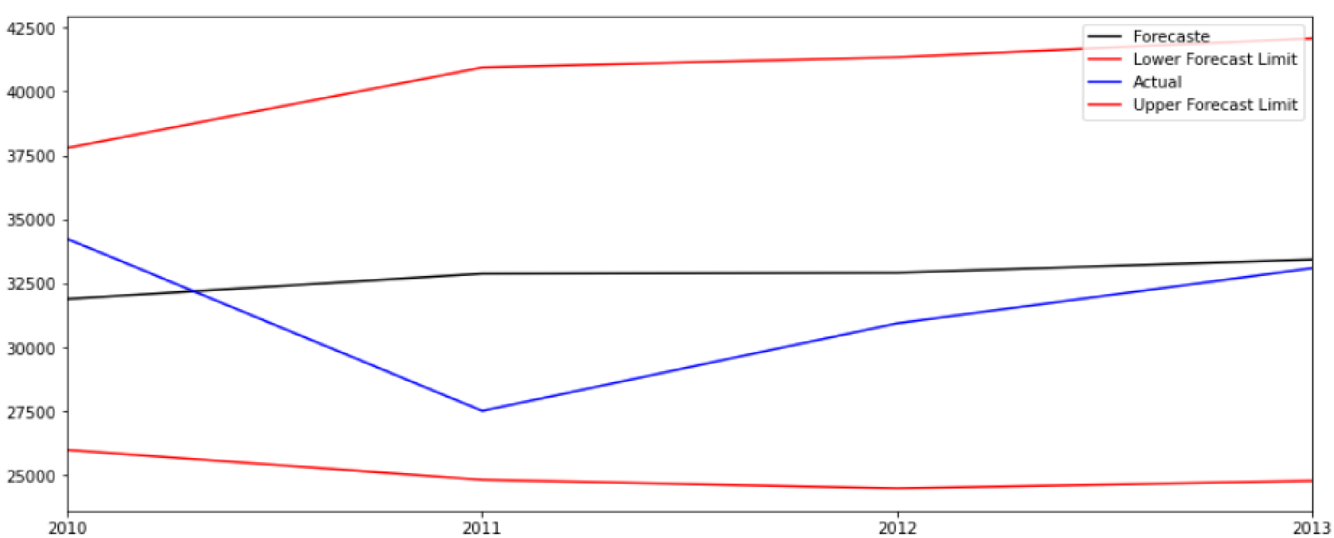
```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                    D.y   No. Observations:                   29
Model:                 ARIMA(2, 1, 1)   Log Likelihood                -273.778
Method:                       css-mle   S.D. of innovations           3013.143
Date:                Fri, 22 Apr 2022   AIC                            557.556
Time:                        10:36:22   BIC                            564.392
Sample:                             1   HQIC                           559.697

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        794.4535    333.770      2.380      0.017     140.276    1448.631
ar.L1.D.y      0.2409      0.353      0.682      0.495      -0.451       0.933
ar.L2.D.y     -0.4942      0.157     -3.148      0.002      -0.802      -0.187
ma.L1.D.y     -0.3123      0.444     -0.704      0.481      -1.182       0.557
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.2437            -1.4014j            1.4224           -0.2226
AR.2            0.2437            +1.4014j            1.4224            0.2226
MA.1            3.2026            +0.0000j            3.2026            0.0000
```

```
steps = 4
fc1, se1, conf1 = model_fit1.forecast(steps)
```

```
result1 =  pd.DataFrame({ 'Forecaste':fc1,
                        'Lower Forecast Limit':conf1[:,0],
                        'Actual':x_test,
                        'Upper Forecast Limit':conf1[:,1]})
result1.index = ts['Number of Immigrants'][30:].index
result1
```

	Forecaste	Lower Forecast Limit	Actual	Upper Forecast Limit
2010-12-31	31884.872968	25979.220254	34235	37790.525682
2011-12-31	32875.026434	24815.769719	27509	40934.283149
2012-12-31	32908.807391	24481.399800	30933	41336.214982
2013-12-31	33423.319384	24774.482961	33087	42072.155806

```
result1.plot(color=['black','red','blue','red'])
plt.show()
```



This is a better model as it fits within the confidence intervals.