DataRitz Technologies
Enhancing Technology Experience

A practical Approach of Data Analytics using Python

A Quick Introduction to Numpy and Pandas

Importing Packages in Python

In [1]:

```
import numpy as np      # Importing Numpy Package (It provides a high-performance multi
dimensional array object)
import pandas as pd     # Importing Pandas Package (It providing high-performance, easy
-to-use data structures
                        # and data analysis tools)
```

numpy array

In [2]:

```
# Creating Numpy array

myArray = np.array([1,2,3,4,5,6])
print(type(myArray))                 # Prints "<class 'numpy.ndarray'> i.e. Type of objec
t"
```

```
<class 'numpy.ndarray'>
```

In [3]:

```
# Creating Two dimension numpy array

twoArray = np.array([[1,1,1],[2,2,2]])
print(type(twoArray))
print(twoArray.shape)  # Print shape of numpy array (2,3) i.e. Two row and three column
s
```

```
<class 'numpy.ndarray'>
(2, 3)
```

In [4]:

```
# Dimension of numpy Array
twoArray.ndim
```

Out[4]:

```
2
```

In [5]:

```
# numpy.arange()
# Returns an array with evenly spaced elements as per the interval.

myRange = np.arange(1,100,3)
myRange
```

Out[5]:

```
array([ 1,  4,  7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49,
       52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97])
```

Numpy Array indexing

> Items in **ndarray** object follows zero-based index

In [6]:

```
arr = np.array([1,23,45,56])
print(arr[0])   #It Prints first element
print(arr[3])   #It Prints fourth element
```

```
1
56
```

Slicing

> Slicing means taking elements from one given index to another given index.

In [7]:

```
# Slice array from index 1 to index 3
arr = np.array([10,20,30,40,3,45])
print(arr[1:4])
```

```
[20 30 40]
```

In [8]:

```
# Slice array from  index 1 to index 4 with alternate element
print(arr[1:5:2])
```

```
[20 40]
```

Boolean array indexing

In [9]:

```
arr = np.arange(1,100)  # create an array range from 1 to 100
arr[arr%2 == 0]
```

Out[9]:

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34,
       36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68,
       70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98])
```

Working with Pandas

Key Features of Pandas

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

DataFrame

> A Data frame is a two-dimensional data structure.

In [10]:

```
#Create a DataFrame
students = pd.DataFrame({'Name':['Ram','Rahul','Rajesh','Sita','Aslam'],
                         'Age':[23,22,24,None,None],
                         'Marks':[80,34,57,65,34],
                         'Branch':['CSE','IT','IT','CSE','CSE']})
students
```

Out[10]:

	Name	Age	Marks	Branch
0	Ram	23.0	80	CSE
1	Rahul	22.0	34	IT
2	Rajesh	24.0	57	IT
3	Sita	NaN	65	CSE
4	Aslam	NaN	34	CSE

In [11]:

```
# Adding new column
students['city']=['Gzb','','','','Goa']
students
```

Out[11]:

	Name	Age	Marks	Branch	city
0	Ram	23.0	80	CSE	Gzb
1	Rahul	22.0	34	IT	
2	Rajesh	24.0	57	IT	
3	Sita	NaN	65	CSE	
4	Aslam	NaN	34	CSE	Goa

In [12]:

```
# Selection of Data

students['Name'] # return column with label 'Name'  as Series
```

Out[12]:

```
0       Ram
1     Rahul
2    Rajesh
3      Sita
4     Aslam
Name: Name, dtype: object
```

In [13]:

```
# Selection of Data
students[['Name','city']]   # returns columns as a new DataFrame
```

Out[13]:

	Name	city
0	Ram	Gzb
1	Rahul	
2	Rajesh	
3	Sita	
4	Aslam	Goa

In [14]:

```
# Selection By postion
students.iloc[0]    # returns index 0 data i.e. row 1
```

Out[14]:

```
Name        Ram
Age          23
Marks        80
Branch      CSE
city        Gzb
Name: 0, dtype: object
```

Filter, Sort and Groupby

In [15]:

```
# Filter the Column
# Display all who got marks greater than 60
students[students['Marks']>60]
```

Out[15]:

	Name	Age	Marks	Branch	city
0	Ram	23.0	80	CSE	Gzb
3	Sita	NaN	65	CSE	

In [16]:

```
# sort values in a certain column in an ascending order
students.sort_values('Name')
```

Out[16]:

	Name	Age	Marks	Branch	city
4	Aslam	NaN	34	CSE	Goa
1	Rahul	22.0	34	IT	
2	Rajesh	24.0	57	IT	
0	Ram	23.0	80	CSE	Gzb
3	Sita	NaN	65	CSE	

In [17]:

```
# Split Data into Groups

grouped=students.groupby('Branch')
grouped
```

Out[17]:

```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x0000016F164C3198
>
```

In [18]:

```
for name,group in grouped:
    print(name)
    print(group)
```

```
CSE
    Name   Age  Marks Branch city
0    Ram  23.0     80    CSE  Gzb
3   Sita   NaN     65    CSE
4  Aslam   NaN     34    CSE  Goa
IT
     Name   Age  Marks Branch city
1   Rahul  22.0     34     IT
2  Rajesh  24.0     57     IT
```

In [19]:

```
grouped['Branch'].count()
```

Out[19]:

```
Branch
CSE    3
IT     2
Name: Branch, dtype: int64
```

In [20]:

```
grouped['Marks'].mean()
```

Out[20]:

```
Branch
CSE    59.666667
IT     45.500000
Name: Marks, dtype: float64
```

Data Cleaning

> Data cleaning is a very important step in data analysis. For example, we always check for missing values in the data.

In [21]:

```
# To check the number of null value
students.isnull().sum()
```

Out[21]:

```
Name      0
Age       2
Marks     0
Branch    0
city      0
dtype: int64
```

In [22]:

```
# replace empty string
students1 = students.replace('', np.nan)
```

In [23]:

```
students1.isnull().sum()
```

Out[23]:

```
Name      0
Age       2
Marks     0
Branch    0
city      3
dtype: int64
```

Handling Missing Value

- Deletion
  - Row
  - Columns
- Imputation
  - Mean, Meadian, Frequency

In [24]:

```
# Age is numerical Data we can either delete or we can fill it with mean of Age
print(students1)
students1['Age'].fillna(students1['Age'].mean(),inplace=True)
print("After Fillna")
print(students1)
```

```
     Name   Age  Marks Branch city
0     Ram  23.0     80    CSE  Gzb
1   Rahul  22.0     34     IT  NaN
2  Rajesh  24.0     57     IT  NaN
3    Sita   NaN     65    CSE  NaN
4   Aslam   NaN     34    CSE  Goa
After Fillna
     Name   Age  Marks Branch city
0     Ram  23.0     80    CSE  Gzb
1   Rahul  22.0     34     IT  NaN
2  Rajesh  24.0     57     IT  NaN
3    Sita  23.0     65    CSE  NaN
4   Aslam  23.0     34    CSE  Goa
```

In [25]:

```
# Drop city

students1.drop(['city'],axis=1,inplace=True)
```

In [26]:

```
students1
```

Out[26]:

	Name	Age	Marks	Branch
0	Ram	23.0	80	CSE
1	Rahul	22.0	34	IT
2	Rajesh	24.0	57	IT
3	Sita	23.0	65	CSE
4	Aslam	23.0	34	CSE

Data Visualization Using Matplotlib

- Matplotlib is one of the most popular python packages used for Data Visualization.

Importing Matplotlib

In [1]:

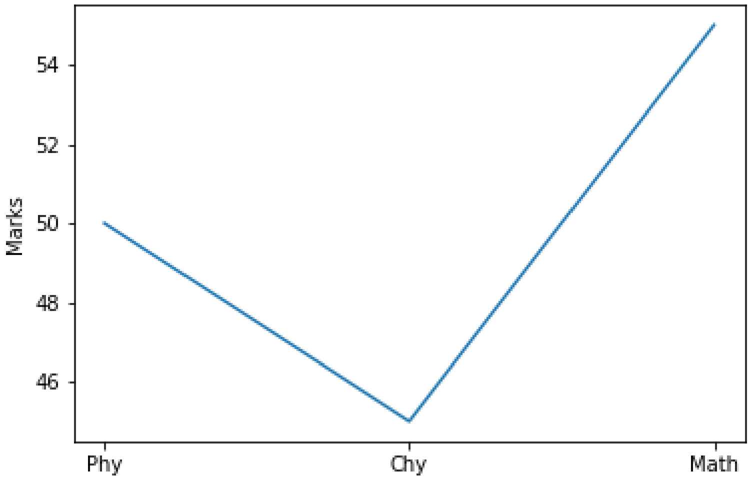
```
from matplotlib import pyplot as plt

# or

import matplotlib.pyplot as plt
```

Plotting using Matplotlib

In [108]:

```
plt.plot(['Phy', 'Chy', 'Math'],[50,45,55])
plt.ylabel('Marks')
plt.show()
```

Out[108]:

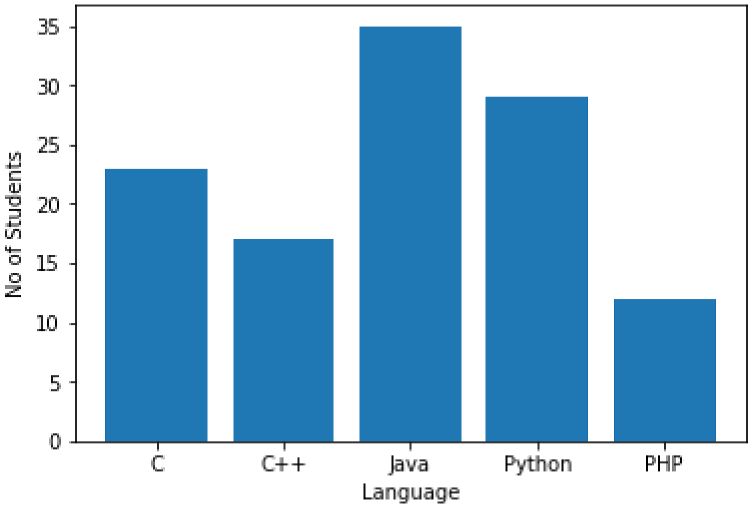


Bar Plot

In [109]:

```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
No_of_students = [23,17,35,29,12]
plt.bar(langs,No_of_students)
plt.xlabel("Language")
plt.ylabel('No of Students')
plt.show()
```

Out[109]:



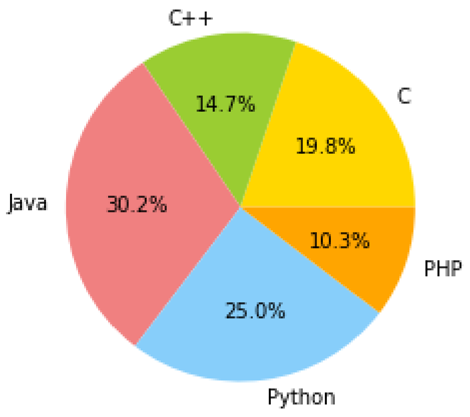
Python Matplotlib : Pie Chart

A pie chart is a circular graph which is broken down into segments. It is generally used to show the percentage or proprotion of each segment.

In [113]:

```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
colors=['gold','yellowgreen','lightcoral','lightskyblue','orange']
plt.pie(students, labels = langs,autopct='%1.1f%%',colors=colors)
plt.show()
```

Out[113]:



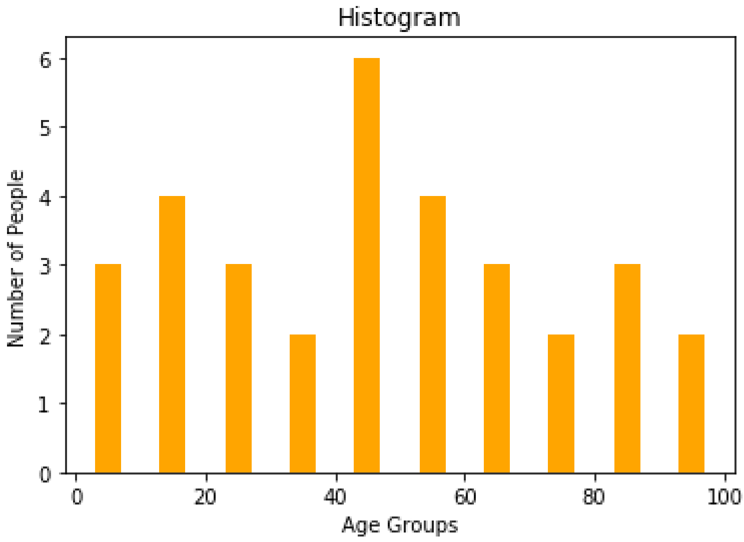
Python Matplotlib : Histogram

Histograms are used to show a distribution whereas a bar chart is used to compare different entities. Histograms are useful when we have data in long arrays. We plot the graph w.r.t the bin. Bin refers to the range of values that are divided into series of intervals

In [110]:

```
import matplotlib.pyplot as plt
age = [12,22,55,62,45,21,22,34,42,37,4,2,102,95,85,55,88,70,95,65,55,7,80,75,65,54,44,43,42,48,11,18,15]
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(age, bins, histtype='bar',rwidth=0.4, color='orange')
plt.xlabel('Age Groups')
plt.ylabel('Number of People')
plt.title('Histogram')
plt.show()
```

Out[110]:



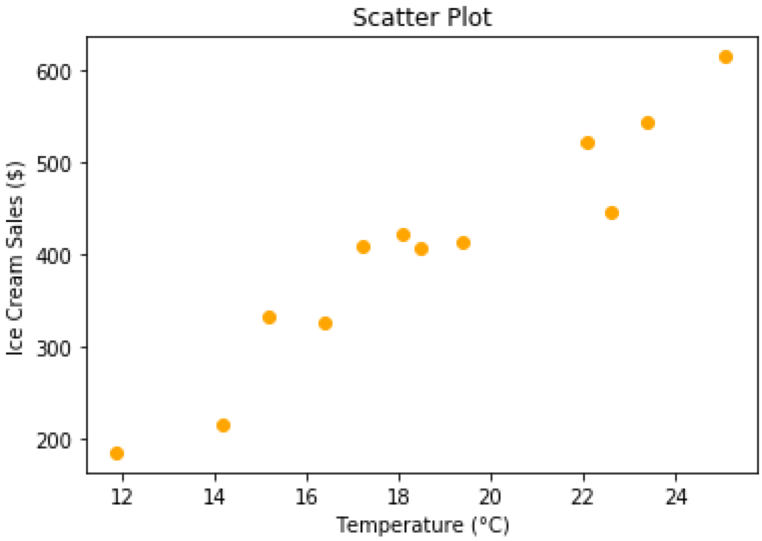
Python Matplotlib : Scatter Plot

Scatter Plot is used in order to compare variables. * For Example, how much one variable is affected by another variable to build a relation out of it. * The data is displayed as a collection of points, each having the value of one variable which determines the position on the horizontal axis and the value of other variable determines the position on the vertical axis.

In [125]:

```
temp = [14.2,16.4,11.9,15.2,18.5,22.1,19.4,25.1,23.4,18.1,22.6,17.2]
sales= [215,325,185,332,406,522,412,614,544,421,445,408]

plt.scatter(temp,sales,color='orange')

plt.xlabel('Temperature (°C)')
plt.ylabel('Ice Cream Sales ($)')
plt.title('Scatter Plot')
plt.show()
```

Out[125]:



Case Study

Covid 19

This dataset has daily level information on the number of affected cases, deaths and recovery from 2019 novel coronavirus.

Row Description

There are total 22189 rows

Column Description

- Sno - Serial number
- ObservationDate - Date of the observation in MM/DD/YYYY
- Province/State - Province or state of the observation (Could be empty when missing)
- Country/Region - Country of observation
- Last Update - Time in UTC at which the row is updated for the given province or country. (Not standardised and so please clean before using it)
- Confirmed - Cumulative number of confirmed cases till that date
- Deaths - Cumulative number of of deaths till that date
- Recovered - Cumulative number of recovered cases till that date

What we want to achieve from this data

- How it is spread wordwide over Time
  - Confirmed Cases over Time
  - Deaths Over Time
  - Recovery Over Time
- Country Wise Analysis over Confirmed, Deaths, and recovery

Importing Library

In [1]:

```
# Importing Library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
```

Importiong DataSet

In [2]:

```
actual_data = pd.read_csv('covid_19_data.csv')
```

In [4]:

```
actual_data.shape
```

Out[4]:

```
(22189, 8)
```

Preprocessing

In [29]:

```
# see top 5 row
actual_data.head()
```

Out[29]:

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths	Rec
0	1	01/22/2020	Anhui	Mainland China	1/22/2020 17:00	1.0	0.0	
1	2	01/22/2020	Beijing	Mainland China	1/22/2020 17:00	14.0	0.0	
2	3	01/22/2020	Chongqing	Mainland China	1/22/2020 17:00	6.0	0.0	
3	4	01/22/2020	Fujian	Mainland China	1/22/2020 17:00	1.0	0.0	
4	5	01/22/2020	Gansu	Mainland China	1/22/2020 17:00	0.0	0.0	

In [30]:

```
# see Bottom 5 row
actual_data.tail()
```

Out[30]:

	SNo	ObservationDate	Province/State	Country/Region	Last Update	Confirmed	Deaths
22184	22185	05/06/2020	Wyoming	US	2020-05-07 02:32:28	631.0	7.0
22185	22186	05/06/2020	Xinjiang	Mainland China	2020-05-07 02:32:28	76.0	3.0
22186	22187	05/06/2020	Yukon	Canada	2020-05-07 02:32:28	11.0	0.0
22187	22188	05/06/2020	Yunnan	Mainland China	2020-05-07 02:32:28	185.0	2.0
22188	22189	05/06/2020	Zhejiang	Mainland China	2020-05-07 02:32:28	1268.0	1.0

In [31]:

```
# Total No of Rows
actual_data.shape
```

Out[31]:

```
(22189, 8)
```

In [32]:

```
# See Info
actual_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22189 entries, 0 to 22188
Data columns (total 8 columns):
SNo                22189 non-null int64
ObservationDate    22189 non-null object
Province/State     10726 non-null object
Country/Region     22189 non-null object
Last Update        22189 non-null object
Confirmed          22189 non-null float64
Deaths             22189 non-null float64
Recovered          22189 non-null float64
dtypes: float64(3), int64(1), object(4)
memory usage: 1.4+ MB
```

In [33]:

```
actual_data.describe()
```

Out[33]:

	SNo	Confirmed	Deaths	Recovered
count	22189.000000	22189.000000	22189.000000	22189.000000
mean	11095.000000	4294.320069	275.478525	1189.001172
std	6405.556897	19481.324990	1852.679828	8214.651165
min	1.000000	0.000000	0.000000	0.000000
25%	5548.000000	13.000000	0.000000	0.000000
50%	11095.000000	138.000000	2.000000	4.000000
75%	16642.000000	981.000000	14.000000	115.000000
max	22189.000000	323978.000000	30076.000000	189910.000000

Performing Data Cleaning

In [34]:

```
# Find null values
actual_data.isna().sum()
```

Out[34]:

```
SNo                    0
ObservationDate        0
Province/State     11463
Country/Region         0
Last Update            0
Confirmed              0
Deaths                 0
Recovered              0
dtype: int64
```

In [35]:

```
# drop Province / State
clean_data = actual_data.drop(['Province/State'],axis=1)
```

In [36]:

```
clean_data.head()
```

Out[36]:

	SNo	ObservationDate	Country/Region	Last Update	Confirmed	Deaths	Recovered
0	1	01/22/2020	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
1	2	01/22/2020	Mainland China	1/22/2020 17:00	14.0	0.0	0.0
2	3	01/22/2020	Mainland China	1/22/2020 17:00	6.0	0.0	0.0
3	4	01/22/2020	Mainland China	1/22/2020 17:00	1.0	0.0	0.0
4	5	01/22/2020	Mainland China	1/22/2020 17:00	0.0	0.0	0.0

Converting Date from string to datetime

In [37]:

```
# Covert date string to date
clean_data['ObservationDate'] = pd.to_datetime(clean_data['ObservationDate'])
```

Lets find some insight in data

Cases over the time WorldWide

In [42]:

```
# Group By ObservationDate
case_over_time = clean_data.groupby(clean_data['ObservationDate'])['Confirmed','Deaths'
,'Recovered'].sum()
case_over_time.head()
```

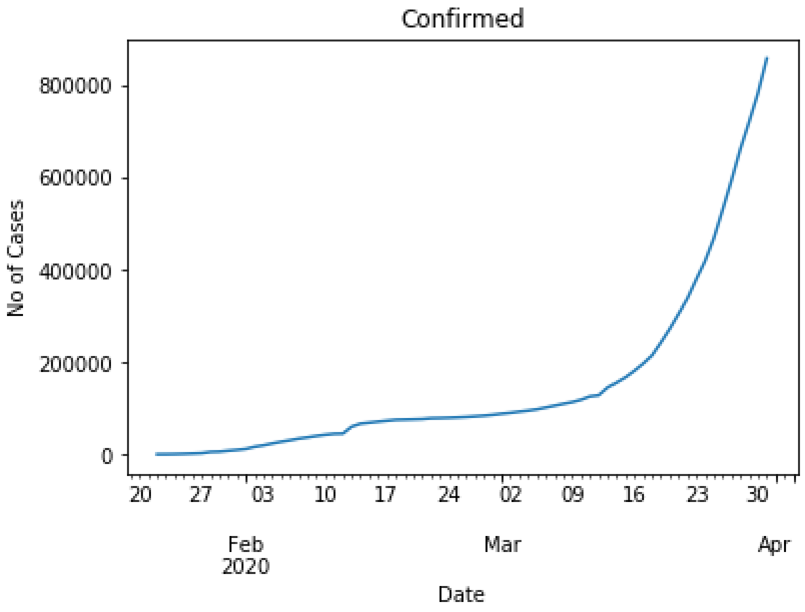
Out[42]:

	Confirmed	Deaths	Recovered
ObservationDate			
2020-01-22	555.0	17.0	28.0
2020-01-23	653.0	18.0	30.0
2020-01-24	941.0	26.0	36.0
2020-01-25	1438.0	42.0	39.0
2020-01-26	2118.0	56.0	52.0

In [66]:

```
# Plot Month vs Confirmed Case
case_over_time.loc['2020-01':'2020-03','Confirmed'].plot()
plt.title("Confirmed")
plt.xlabel('Date')
plt.ylabel('No of Cases')
plt.show()
```

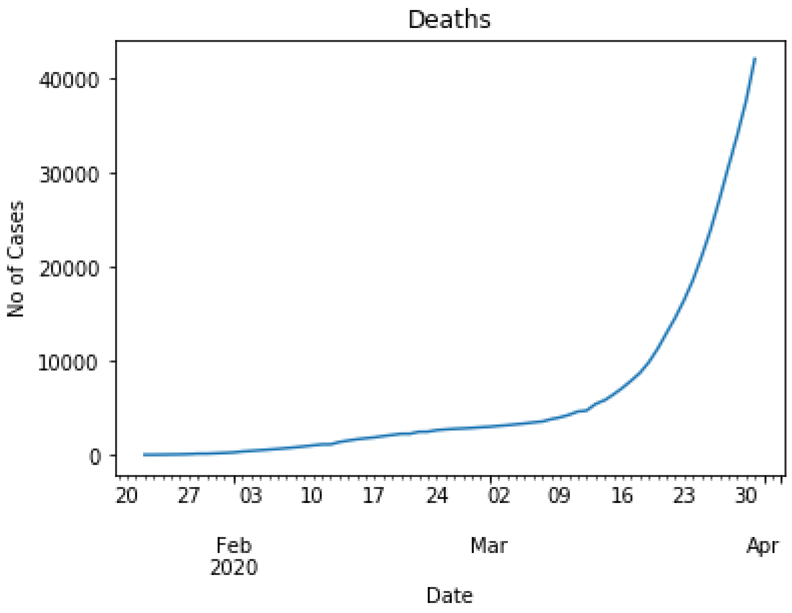
Out[66]:



In [67]:

```
# Plot Month vs Deaths Case
case_over_time.loc['2020-01':'2020-03','Deaths'].plot()
plt.title("Deaths")
plt.xlabel('Date')
plt.ylabel('No of Cases')
plt.show()
```

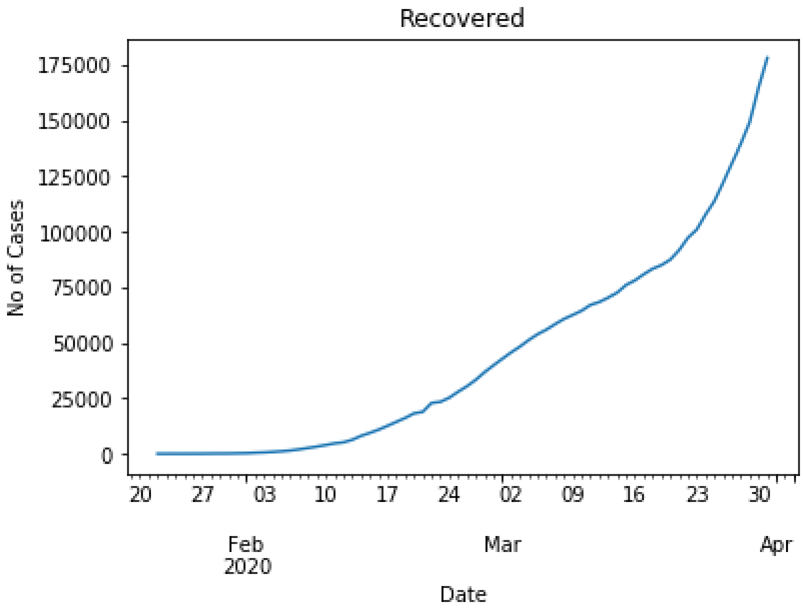
Out[67]:



In [68]:

```
# Plot Month vs Recovered Case
case_over_time.loc['2020-01':'2020-03','Recovered'].plot()
plt.title("Recovered")
plt.xlabel('Date')
plt.ylabel('No of Cases')
plt.show()
```

Out[68]:



Country wise Cases

In [70]:

```
country_wise = clean_data.groupby(clean_data['Country/Region'])['Confirmed','Deaths','R
ecovered'].sum()
country_wise.tail()
```

Out[70]:

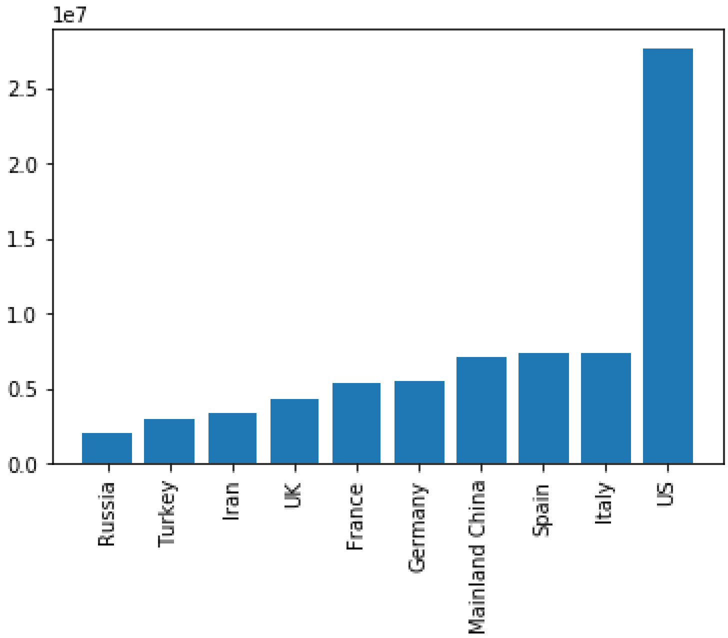
	Confirmed	Deaths	Recovered
Country/Region			
Western Sahara	178.0	0.0	70.0
Yemen	117.0	19.0	13.0
Zambia	2721.0	81.0	1298.0
Zimbabwe	881.0	119.0	72.0
occupied Palestinian territory	25.0	0.0	0.0

Top Ten Country

In [75]:

```
# Top Ten Country Confermed Case
country_wise.sort_values('Confirmed',inplace=True)
top10 = country_wise.tail(10)

plt.bar(top10.index,top10['Confirmed'])
plt.xticks(rotation=90)
plt.show()
```

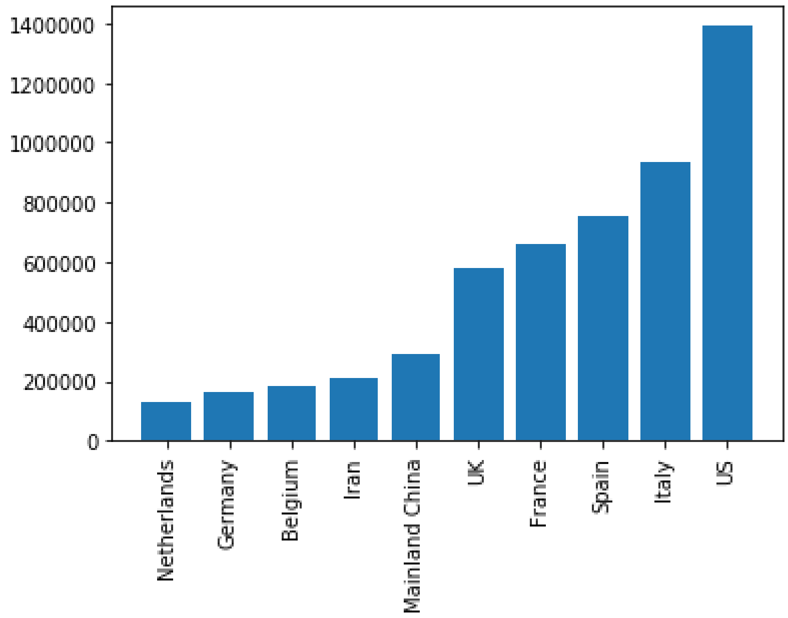
Out[75]:



In [166]:

```
# Top Ten Country Death Case
T_deaths = country_wise.sort_values('Deaths')
top10 = T_deaths.tail(10)

plt.bar(top10.index,top10['Deaths'])
plt.xticks(rotation=90)
plt.show()
```

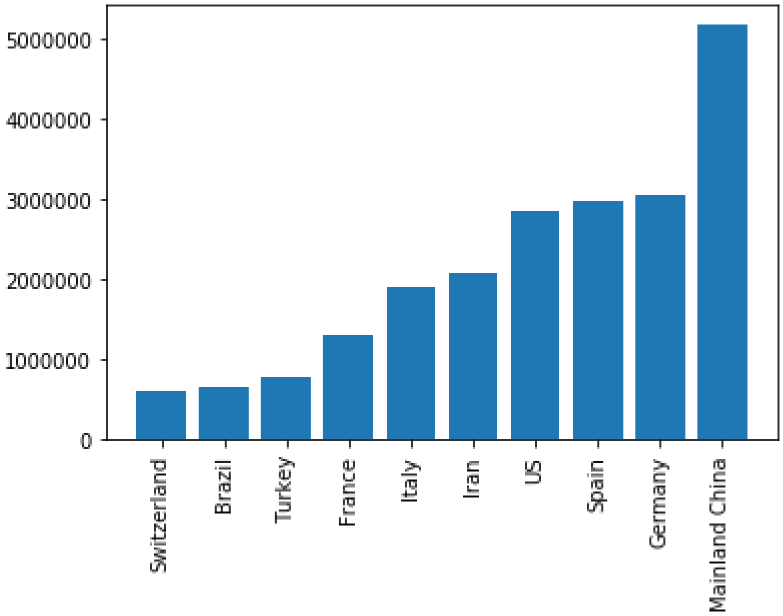
Out[166]:



In [176]:

```
# Top Ten Country Recovered Case
T_Reco = country_wise.sort_values('Recovered')
top10 = T_Reco.tail(10)

plt.bar(top10.index,top10['Recovered'])
plt.xticks(rotation=90)
plt.show()
```

Out[176]:

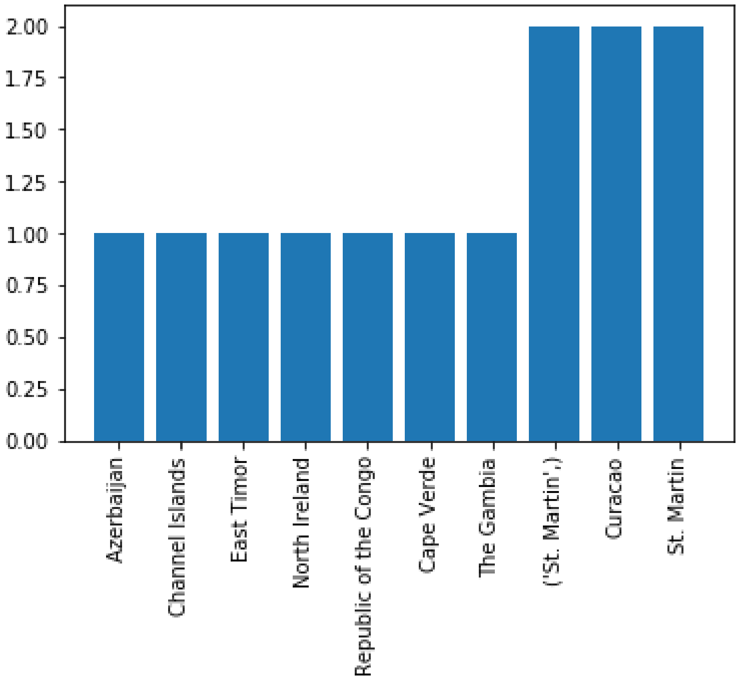


Bottom Ten Country

In [167]:

```
b10 = country_wise.head(10)

plt.bar(b10.index,b10['Confirmed'])
plt.xticks(rotation=90)
plt.show()
```

Out[167]:



World Updates

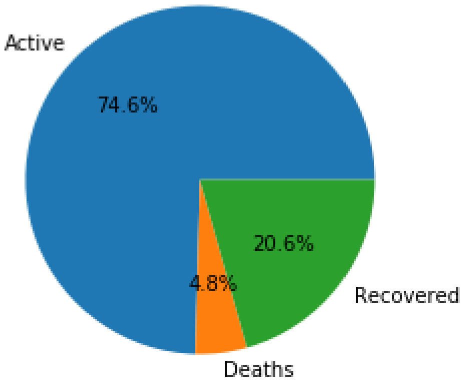
In [168]:

```
Total_conf = country_wise['Confirmed'].sum()
Total_death = country_wise['Deaths'].sum()
Total_recov = country_wise['Recovered'].sum()
Total_active = Total_conf-(Total_death+Total_recov)
```

In [169]:

```
y=[Total_active,Total_death,Total_recov]
plt.pie(cases, labels = label,autopct='%1.1f%%')
plt.show()
```

Out[169]:



India Updates

In [170]:

```
India_case=country_wise.loc['India']
India_case
```

Out[170]:

```
Confirmed    726230.0
Deaths        23624.0
Recovered    161596.0
Name: India, dtype: float64
```

In [171]:

```
India_active = India_case[0]-(India_case[1]+India_case[2])
```

In [172]:

```
y=[India_active,India_case[1],India_case[2]]
plt.pie(y,autopct='%1.1f%%')
plt.show()
```

Out[172]: