Practical 1:

AIM: Python program to display details about the operating system, working directory, files And directories in the current directory, lists the files and all directories, scan and classify them as directories and files

```
import os
def scan_and_classify(path='.'):
    directories = []
    files = []
    for entry in os.listdir(path):
        full_path = os.path.join(path, entry)
        if os.path.isdir(full_path):
            directories.append(entry)
        elif os.path.isfile(full_path):
            files.append(entry)
    return directories, files
path = '.'
directories, files = scan_and_classify(path)
print("Directories:")
for directory in directories:
    print(directory)
print("\nFiles:")
for file in files:
    print(file)
```

Output:

```
Directories:
.ipynb_checkpoints
OS module

Files:
array.ipynb
array.py
Array_to_machine_values.ipynb
Calculator.py
Coin flip.ipynb
l1.py
lab.py
os modules.ipynb
Pandas.ipynb
Question.txt
Random Array.ipynb
Rock Paper Scissor.ipynb
string_symmetry.py
unit_1.txt
Untitled.ipynb
Untitled1.ipynb
Untitled2.ipynb
```

Practical 2:

AIM: Python program to convert an array to an array of machine values and vice versa

```
import struct
def array_to_machine_values(arr, format_char):
    return struct.pack(f'{len(arr)}{format_char}', *arr)
def machine_values_to_array(machine_values, format_char):
    num_elements = len(machine_values) // struct.calcsize(format_char)
    return list(struct.unpack(f'{num_elements}{format_char}', machine_values))
original_array = [10, 20, 30, 40]
format_char = 'i' # 'i' is for integer
machine_values = array_to_machine_values(original_array, format_char)
print(f"Machine values (bytes): {machine_values}")
restored_array = machine_values_to_array(machine_values, format_char)
print(f"Restored array: {restored_array}")
```

Output:

```
Machine values (bytes): b'\n\x00\x00\x00\x14\x00\x00\x00\x1e\x00\x00\x00(\x00\x00\x00'
Restored array: [10, 20, 30, 40]
```

Practical 3:

AIM: Python program to get information about the file pertaining to the file mode and to get time values with components using local time and gm time.

```
import os
import time
def get_file_info(file_path):
    file_stat = os.stat(file_path)
    file_mode = oct(file_stat.st_mode)  # Convert to octal string representation
    access_time = file_stat.st_atime  # Time of last access
    modify_time = file_stat.st_mtime  # Time of last modification
    change_time = file_stat.st_ctime  # Time of last status change (creation time on Windows)
    access_time_local = time.localtime(access_time)
    modify_time_local = time.localtime(modify_time)
    change_time_local = time.localtime(change_time)
    access_time_gmt = time.gmtime(access_time)
    modify_time_gmt = time.gmtime(modify_time)
    change_time_gmt = time.gmtime(change_time)
    file_info = {
        "file_mode": file_mode,
        "access_time_local": time.strftime('%Y-%m-%d %H:%M:%S', access_time_local),
        "modify_time_local": time.strftime('%Y-%m-%d %H:%M:%S', modify_time_local),
        "change_time_local": time.strftime('%Y-%m-%d %H:%M:%S', change_time_local),
        "access_time_gmt": time.strftime('%Y-%m-%d %H:%M:%S', access_time_gmt),
        "modify_time_gmt": time.strftime('%Y-%m-%d %H:%M:%S', modify_time_gmt),
        "change_time_gmt": time.strftime('%Y-%m-%d %H:%M:%S', change_time_gmt),
    }
    return file_info
```

```
file_path = "Untitled2.ipynb"
file_info = get_file_info(file_path)
print(f"File Mode: {file_info['file_mode']}")
print(f"Access Time (Local): {file_info['access_time_local']}")
print(f"Modify Time (Local): {file_info['modify_time_local']}")
print(f"Change Time (Local): {file_info['change_time_local']}")
print(f"Access Time (GMT): {file_info['access_time_gmt']}")
print(f"Modify Time (GMT): {file_info['modify_time_gmt']}")
print(f"Change Time (GMT): {file_info['change_time_gmt']}")
```

Output:

```
File Mode: 0o100666
Access Time (Local): 2024-09-19 20:30:25
Modify Time (Local): 2024-09-19 20:30:25
Change Time (Local): 2024-09-19 20:08:24
Access Time (GMT): 2024-09-19 15:00:25
Modify Time (GMT): 2024-09-19 15:00:25
Change Time (GMT): 2024-09-19 14:38:24
```

Practical 4:

AIM: Python program to connect to Google using socket programming

```
import socket
def connect_to_google():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = 'www.google.com'
    port = 80  # HTTP port
   try:
        remote_ip = socket.gethostbyname(host)
        print(f"IP address of {host}: {remote_ip}")
        s.connect((remote_ip, port))
        print(f"Successfully connected to {host} on port {port}")
        request = "GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n"
        s.send(request.encode())
        response = b""
        while True:
            part = s.recv(4096)
            if not part:
                break
            response += part
        print(f"Received response from {host}:\n{response.decode('utf-8', errors='ignore')}")
    except socket.error as e:
         print(f"Socket error: {e}")
    finally:
        s.close()
# Run the example
connect_to_google()
```

Output:

```
IP address of www.google.com: 142.250.206.132
Successfully connected to www.google.com on port 80
Received response from www.google.com:
HTTP/1.1 200 OK
Date: Thu, 19 Sep 2024 15:18:54 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-uuclUEABB0YXs97vNmsang' 'strict-dynamic' 'report-sample' 'unsaf
-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: AEC=AVYB7cqk9cWUaJ1eJPlBr_kvdmEycxYg4iAZy7ddS7mAiDBcdaPhZoPOzHU; expires=Tue, 18-Mar-2025 15:18:54 GMT; path=/; domain=.google.com; Secure;
ttpOnly; SameSite=lax
Set-Cookie: NID=517=YJcUaZZWPIWLoFCdUQ2ik4eGGze6wW__FdoTZEbxPz9UAfG1CT7k0fp8GExfM60jcjG_wFZoxOdOam-kdM9R2CrPK6h8_7uhuoaiIjhjJS_fCuO02nVN5dseLXwcFKGIM28T
M48byY_F0pTCmzozEEJ-cH5W5gDNCgnrpLpmQxc9d2Yhz1QlsGP44VzFyn43F1A; expires=Fri, 21-Mar-2025 15:18:54 GMT; path=/; domain=.google.com; HttpOnly
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked

5e5d
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-T
pe"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="uuclUEABB0YXs97vNm
ang">(function(){var _g={kEI:'3kDsZpOQEMXP1e8P8sb52A8',kEXPI:'0,3700262,687,432,3,48,448480,93005,2891,89155,18161,162437,23024,6699,41946,57737,2,2,1,2
632,8155,23350,22436,9779,62657,6050,27515,42644,15816,1804,7734,40983,13493,15783,11106,3075,1490,6109,5303,5213674,146,3,56,5991227,2841670,109,279803
6,16672,43887,3,1603,3,2124363,23029351,8163,10336,10736,2728,147,58705,22465,11650,10973,15164,8182,49429,21675,6749,155,2,2482,13416,5294,2530,9138,74
0,2,2,3856,328,3217,4,1238,1766,1117,19484,4858,5,821,36,1669,5633,687,2749,2,5045,3,5,3010,2695,4479,1892,24,4167,5074,710,682,283,168,213,5844,2005,57
7,56,353,1860,2,9,739,4,2350,2121,6,3,3296,39,716,4134,2379,1484,981,528,2767,9672,4826,1342,685,1539,1712,2464,797,2545,489,2749,2900,1,2642,573,1935,2
0,2086,156,4,3,567,2,6485,625,3179,376,4144,287,2,3,126,2,3281,831,1344,883,1326,2,304,256,597,1801,301,200,14,399,120,266,14,646,612,755,3,2,1015,25,80
580,830,293,2740,250,3668,224,2,2,3,1451,41,685,1228,4,2,3,640,497,230,2,2249,81,1018,14,2,293,1220,2,46,299,217,5,3144,1358,257,126,65,2,76,2,1827,123,
2,471,259,372,12,900,47,4,253,1025,192,219,597,621,4,94,402,3,170,124,59,162,1346,607,54,595,105,62,92,468,146,908,40,40,390,430,516,2,145,1,771,33,135,
39,140,338,508,2,5,2,431,4,1,6,173,1549,132,144,155,116,323,74,262,2,45,1,3,7,232,16,614,66,1465,49,75,346,119,389,556,9,389,1056,601,1,6,85,1017,2399,1
9,5,425,17,1324,342,2,21455626,3,1423,3,14475,3,16997,18,150,438,2685,983275',kBL:'5q5S',kOPI:89978449};(function(){var a;((a=window.google)==null?0:a.s
vsc)?google.kEI=_g.kEI:window.google=_g;}).call(this);})();(function(){google.sn='webhp';google.kHL='en-IN';})();(function(){
var h=this||self;function l(){return window.google!==void 0&&window.google.kOPI!==void 0&&window.google.kOPI!==0?window.google.kOPI:null};var m,n=[];fun
tion p(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid"))));)a=a.parentNode;return b||m}function q(a){for(var b=null;a&&(!a.getAttribute||!(b=a
getAttribute("leid"))));)a=a.parentNode;return b}function r(a){/^http:/i.test(a)&&window.location.protocol==="https:"&&(google.ml&&google.ml(Error("a"),
```

Practical 5

AIM: Python program to perform Array operations using Numpy package

```
import numpy as np


def array_operations():
    arr1 = np.array([10, 20, 30, 40])
    arr2 = np.array([50, 60, 70, 80])
    print("Array 1:", arr1)
    print("Array 2:", arr2)
    arr_add = np.add(arr1, arr2)
    print("\nAddition of arrays:", arr_add)
    arr_sub = np.subtract(arr2, arr1)
    print("Subtraction of arrays:", arr_sub)
    arr_mul = np.multiply(arr1, arr2)
    print("Multiplication of arrays:", arr_mul)
    arr_div = np.divide(arr2, arr1)
    print("Division of arrays:", arr_div)
    arr_dot = np.dot(arr1, arr2)
    print("\nDot product of arrays:", arr_dot)
    arr_concat = np.concatenate((arr1, arr2))
    print("\nConcatenation of arrays:", arr_concat)
    arr_reshaped = np.reshape(arr_concat, (2, 4))
    print("\nReshaped array (2x4):\n", arr_reshaped)
    arr_transposed = np.transpose(arr_reshaped)
    print("\nTransposed array (4x2):\n", arr_transposed)
    max_element = np.max(arr1)
    print("\nMaximum element in Array 1:", max_element)
    min_element = np.min(arr2)
    print("Minimum element in Array 2:", min_element)
```

```
    sum_elements = np.sum(arr1)
    print("Sum of elements in Array 1:", sum_elements)
    mean_elements = np.mean(arr2)
    print("Mean of elements in Array 2:", mean_elements)
    arr_slice = arr_concat[2:6]
    print("\nSliced array (from index 2 to 5):", arr_slice)
array_operations()
```

Output:

```
Array 1: [10 20 30 40]
Array 2: [50 60 70 80]

Addition of arrays: [ 60  80 100 120]
Subtraction of arrays: [40 40 40 40]
Multiplication of arrays: [ 500 1200 2100 3200]
Division of arrays: [5.         3.         2.33333333 2.        ]

Dot product of arrays: 7000

Concatenation of arrays: [10 20 30 40 50 60 70 80]

Reshaped array (2x4):
 [[10 20 30 40]
 [50 60 70 80]]

Transposed array (4x2):
 [[10 50]
 [20 60]
 [30 70]
 [40 80]]

Maximum element in Array 1: 40
Minimum element in Array 2: 50
Sum of elements in Array 1: 100
Mean of elements in Array 2: 65.0

Sliced array (from index 2 to 5): [30 40 50 60]
```

Practical 6

AIM: Python program to perform Data Manipulation operations using Pandas package.

```
import pandas as pd
import numpy as np

data = {
    'Name': ['Mayank', 'Nilesh', 'Taniya', 'Suhani', 'Rakesh'],
    'Age': [20, np.nan, 20, 19, 46],
    'City': ['Dwarka', 'Gurugram', 'Delhi', 'Chennai', 'Farrukhnagar'],
    'Salary': [90000, 80000, np.nan, 60000, 75000]
}

df = pd.DataFrame(data)

# 2. Basic DataFrame Operations
print("DataFrame:")
print(df)
print("\nFirst 3 rows:")
print(df.head(3))
print("\nDataFrame info:")
print(df.info())
print("\nDescriptive statistics:")
print(df.describe())

# 3. Indexing and Selection
print("\nSelecting 'Name' and 'City':")
print(df[['Name', 'City']])
```

```
# 4. Data Cleaning
print("\nDropping rows with missing values:")
df_cleaned = df.dropna()
print(df_cleaned)
print("\nFilling missing 'Age' with mean:")
df['Age'].fillna(df['Age'].mean(), inplace=True)
print(df)

# 5. Data Transformation
print("\nSorting by 'Salary':")
df_sorted = df.sort_values(by='Salary')
print(df_sorted)
print("\nGrouping by 'City' and calculating mean salary:")
print(df.groupby('City')['Salary'].mean())

# 6. Adding/Removing Columns
df['Experience'] = [2, 3, 1, 5, 4]  # Adding a new column
print("\nDataFrame after adding 'Experience':")
print(df)
df.drop(columns=['Experience'], inplace=True)  # Removing the 'Experience' column
print("\nDataFrame after dropping 'Experience':")
print(df)

# 7. Filtering and Boolean Indexing
print("\nFiltering for Age > 25:")
filtered_df = df[df['Age'] > 25]
print(filtered_df)

# 8. Merging and Joining
```

```
data2 = {
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix'],
    'State': ['NY', 'CA', 'IL', 'TX', 'AZ']
}
df2 = pd.DataFrame(data2)
print("\nMerging DataFrames on 'City':")
merged_df = pd.merge(df, df2, on='City')
print(merged_df)

# 9. Time Series
date_range = pd.date_range(start='2024-01-01', periods=5)
time_df = pd.DataFrame({'Date': date_range, 'Value': [10, 20, 30, 40, 50]})
time_df.set_index('Date', inplace=True)
print("\nTime Series DataFrame:")
print(time_df)
print("\nResampling to get monthly sum:")
print(time_df.resample('M').sum())

# 10. Exporting Data
df.to_csv('output.csv', index=False)
print("\nDataFrame exported to 'output.csv'.")
```

Output:

```
Selecting 'Name' and 'City':
     Name           City
0  Mayank         Dwarka
1  Nilesh      Gurugram
2  Taniya          Delhi
3  Suhani        Chennai
4  Rakesh  Farrukhnagar

Dropping rows with missing values:
     Name   Age           City   Salary
0  Mayank  20.0         Dwarka  90000.0
3  Suhani  19.0        Chennai  60000.0
4  Rakesh  46.0  Farrukhnagar  75000.0

Filling missing 'Age' with mean:
     Name    Age           City   Salary
0  Mayank  20.00         Dwarka  90000.0
1  Nilesh  26.25       Gurugram  80000.0
2  Taniya  20.00          Delhi      NaN
3  Suhani  19.00        Chennai  60000.0
4  Rakesh  46.00  Farrukhnagar  75000.0

Sorting by 'Salary':
     Name    Age           City   Salary
3  Suhani  19.00        Chennai  60000.0
4  Rakesh  46.00  Farrukhnagar  75000.0
1  Nilesh  26.25       Gurugram  80000.0
0  Mayank  20.00         Dwarka  90000.0
2  Taniya  20.00          Delhi      NaN

Grouping by 'City' and calculating mean salary:
City
Chennai          60000.0
Delhi                NaN
Dwarka           90000.0
Farrukhnagar     75000.0
Gurugram         80000.0
Name: Salary, dtype: float64
DataFrame:
     Name   Age           City   Salary
0  Mayank  20.0         Dwarka  90000.0
1  Nilesh   NaN       Gurugram  80000.0
2  Taniya  20.0          Delhi      NaN
3  Suhani  19.0        Chennai  60000.0
4  Rakesh  46.0  Farrukhnagar  75000.0

First 3 rows:
     Name   Age       City   Salary
0  Mayank  20.0     Dwarka  90000.0
1  Nilesh   NaN   Gurugram  80000.0
2  Taniya  20.0      Delhi      NaN

DataFrame info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    5 non-null      object
 1   Age     4 non-null      float64
 2   City    5 non-null      object
 3   Salary  4 non-null      float64
dtypes: float64(2), object(2)
memory usage: 292.0+ bytes
None

Descriptive statistics:
             Age   Salary
count   4.000000      4.0
mean   26.250000  76250.0
std    13.175103  12500.0
min    19.000000  60000.0
25%    19.750000  71250.0
50%    20.000000  77500.0
75%    26.500000  82500.0
max    46.000000  90000.0
```

```
     Name    Age          City   Salary  Experience
0  Mayank  20.00        Dwarka  90000.0           2
1  Nilesh  26.25      Gurugram  80000.0           3
2  Taniya  20.00         Delhi      NaN           1
3  Suhani  19.00       Chennai  60000.0           5
4  Rakesh  46.00  Farrukhnagar  75000.0           4

DataFrame after dropping 'Experience':
     Name    Age          City   Salary
0  Mayank  20.00        Dwarka  90000.0
1  Nilesh  26.25      Gurugram  80000.0
2  Taniya  20.00         Delhi      NaN
3  Suhani  19.00       Chennai  60000.0
4  Rakesh  46.00  Farrukhnagar  75000.0

Filtering for Age > 25:
     Name    Age          City   Salary
1  Nilesh  26.25      Gurugram  80000.0
4  Rakesh  46.00  Farrukhnagar  75000.0

Merging DataFrames on 'City':
Empty DataFrame
Columns: [Name, Age, City, Salary, State]
Index: []

Time Series DataFrame:
            Value
Date
2024-01-01     10
2024-01-02     20
2024-01-03     30
2024-01-04     40
2024-01-05     50

Resampling to get monthly sum:
            Value
Date
2024-01-31    150

DataFrame exported to 'output.csv'.
```

Practical 7

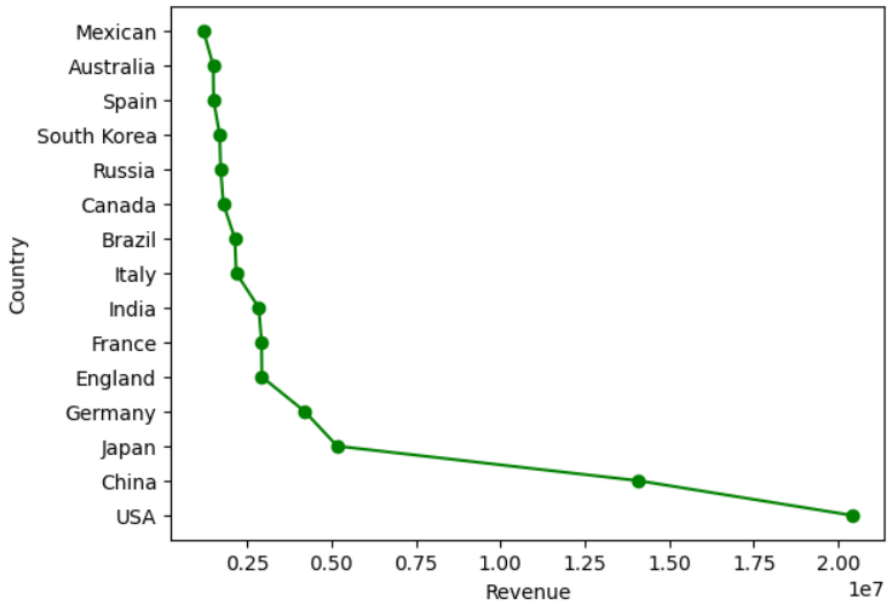
AIM: Python program to display multiple types of charts using Matplotlib package.

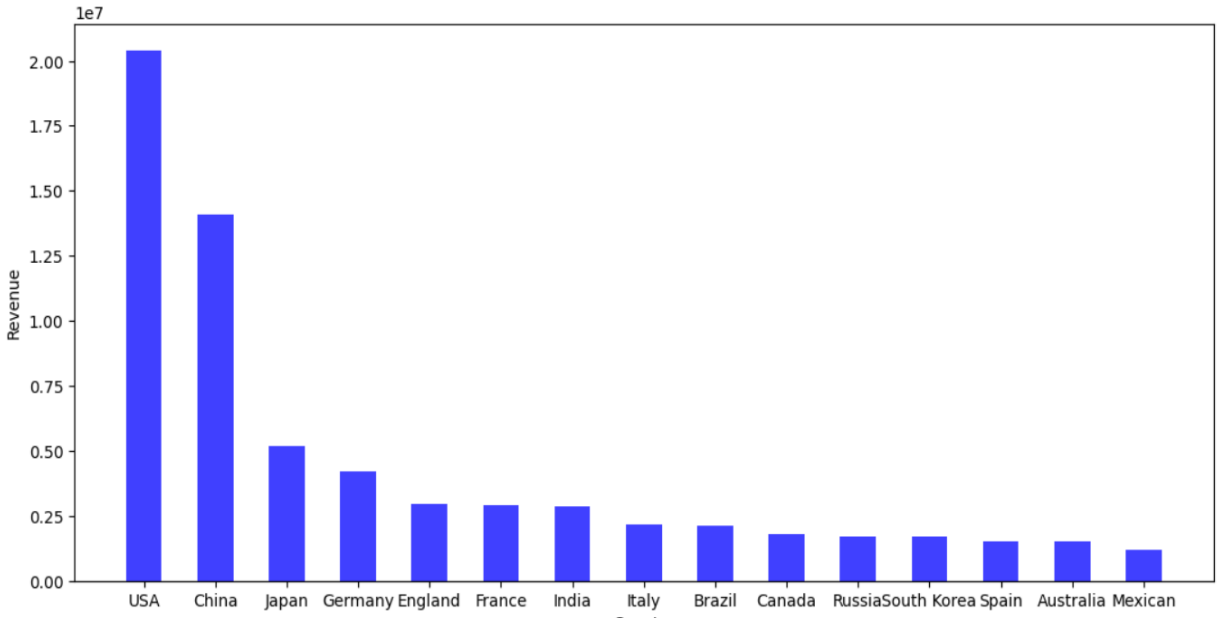
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data=pd.read_csv(r"D:\Programming\Datasets\Dataset_1\dataset_1.csv")
top_15=data.sort_values(by='Revenue',ascending=False).head(15)
print("The top 15 sorted data of csv file : \n \n",top_15)
#Line plot
plt.plot(top_15['Revenue'],top_15['Country'],marker="o",color="Green")
plt.xlabel("Revenue")
plt.ylabel("Country")
#Bar Plot
plt.bar(top_15['Country'],top_15['Revenue'],width=0.5,color="Blue",alpha=0.75)
plt.xlabel("Country")
plt.ylabel("Revenue")
#Scatter Plot
plt.scatter(top_15['Revenue'],top_15['Country'],marker="o",color="Green")
plt.xlabel("Revenue")
plt.ylabel("Country")
#Histogram plot
plt.hist(top_15['Revenue'],bins=15,color="cyan",edgecolor="Red")
#Pie plot
plt.pie(top_15['Revenue'],labels=top_15['Country'],autopct='%1.1f')
#Area plot
plt.stackplot(top_15['Revenue'],top_15['Country'],color="Green",alpha=0.5)
plt.xlabel("Revenue")
plt.ylabel("Country")
#KDE plot
```

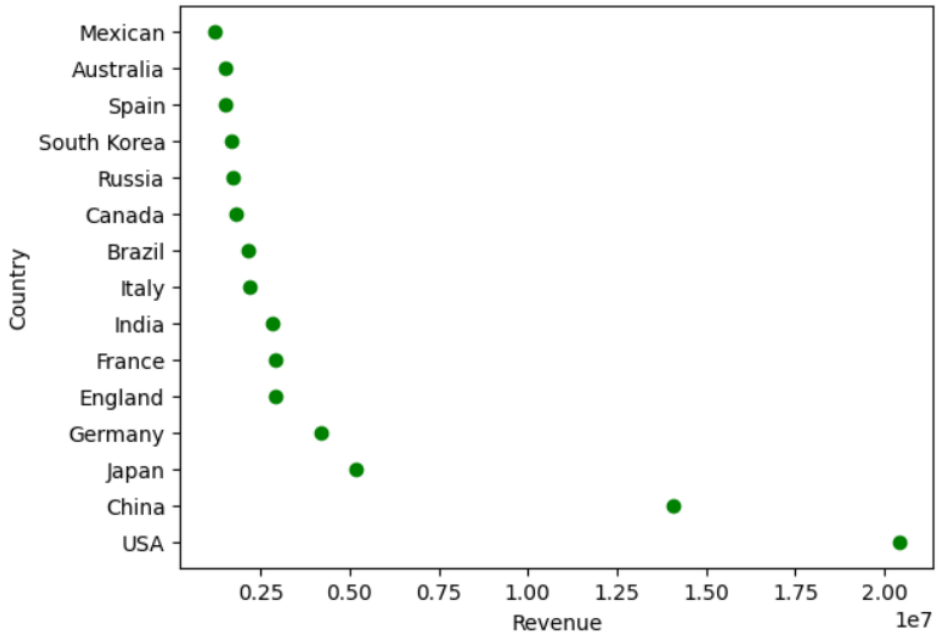
```
sns.kdeplot(top_15['Revenue'], fill=True, color="Blue", alpha=0.5,bw_adjust=0.5)
plt.show()
```

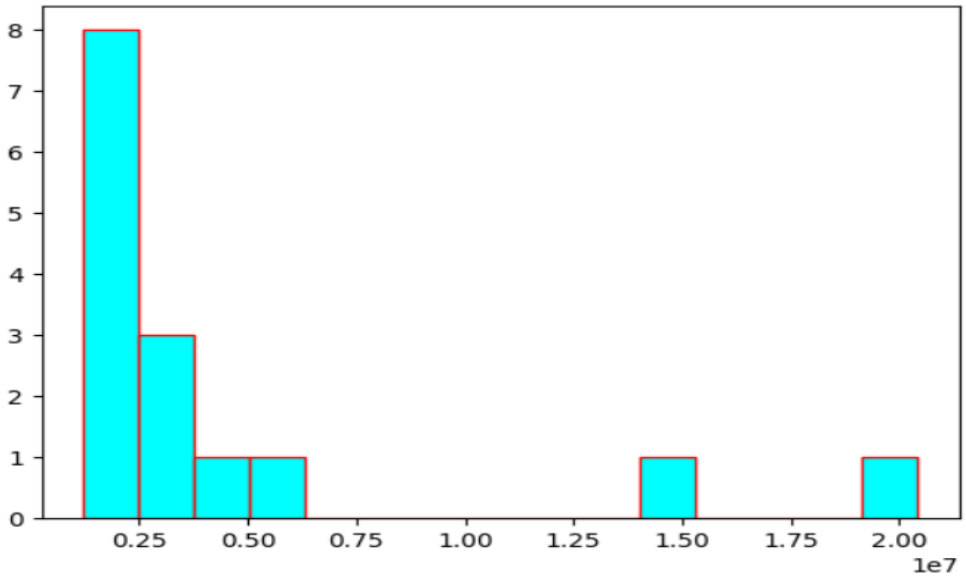
Output:

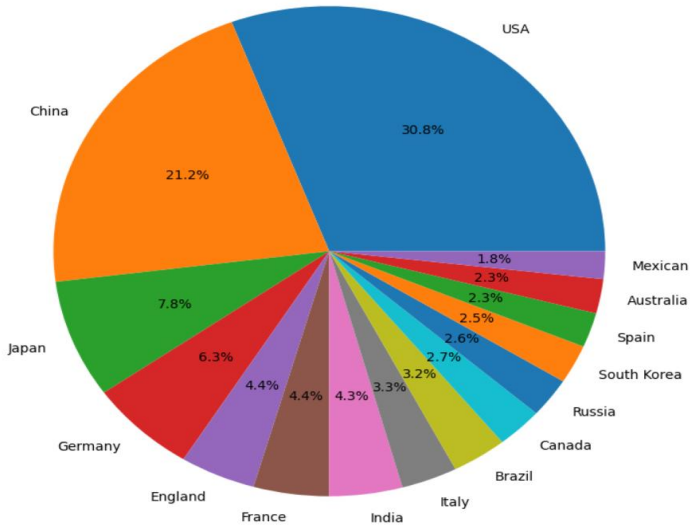
```
The top 15 sorted data of csv file :

          Country    Revenue     Nüfus   Per person       Continent
0             USA   20410000    323766        63039   North America
1           China   14090000   1415045         9957            Asia
2           Japan    5170000    127185        40649            Asia
3         Germany    4210000     82293        51158          Europe
4         England    2940000     66573        44162          Europe
5          France    2930000     65233        44915          Europe
6           India    2850000   1354051         2104            Asia
7           Italy    2180000     62273        36768          Europe
8          Brazil    2140000    210867        10148   South America
9          Canada    1800000     36953        48710   North America
10         Russia    1720000    143967        11947          Europe
11   South Korea     1690000     51164        33031            Asia
12          Spain    1510000     46397        32545          Europe
13     Australia    1500000     24772        60552         Oceania
14        Mexican   1210000    130759         9253   North America
```

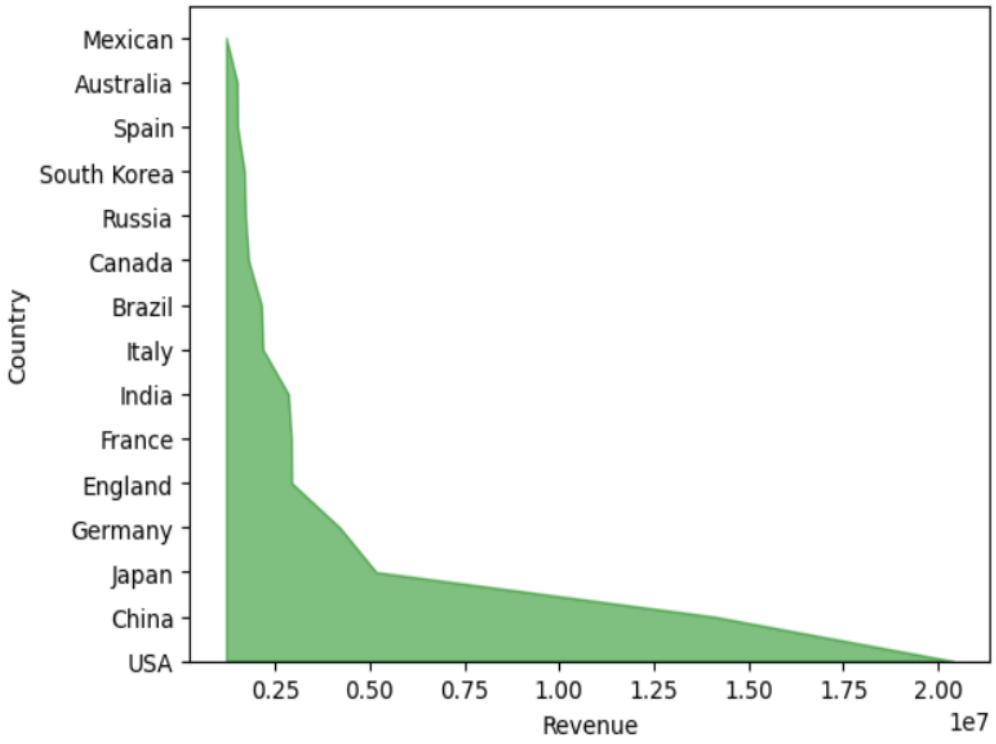


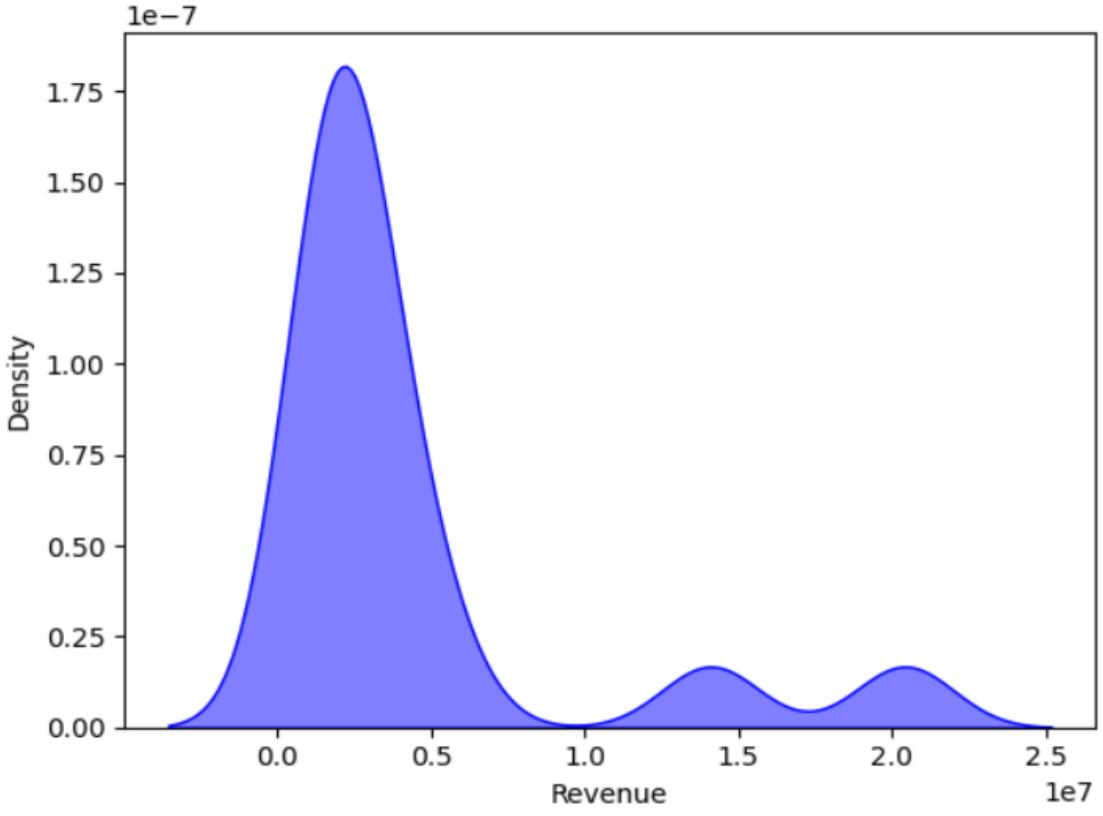



Mexican
Australia
Spain
South Korea
Russia
Canada
Brazil
Italy
India
France
England
Germany
Japan
China
USA
Country
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7
Revenue


8
7
6
5
4
3
2
1
0
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7


USA
30.8%
China
21.2%
Japan
7.8%
Germany
6.3%
England
4.4%
France
4.4%
India
4.3%
Italy
3.3%
Brazil
3.2%
Canada
2.7%
Russia
2.6%
South Korea
2.5%
Spain
2.3%
Australia
2.3%
Mexican
1.8%


Mexican
Australia
Spain
South Korea
Russia
Canada
Brazil
Italy
India
France
England
Germany
Japan
China
USA
Country
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e7
Revenue


1e−7
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
0.0
0.5
1.0
1.5
2.0
2.5
1e7
Revenue