# Artificial Intelligence with Python

## Lab Report 03: PANDAS

- Name: Ghulam Sarwar
- CMS ID: 033-16-0017

**Instructor: Dr. Gulsher Ali**

1- Load the Titanic Dataset in Kaggle notebook from given link below: https://www.kaggle.com/hesh97/titanicdataset-traincsv (https://www.kaggle.com/hesh97/titanicdataset-traincsv) Use the bar plot to show following plots:

*PART_01*

**Plot the Number of people survived and did not survive. Hint: Plot the counts values of “Survived” column of dataset.**

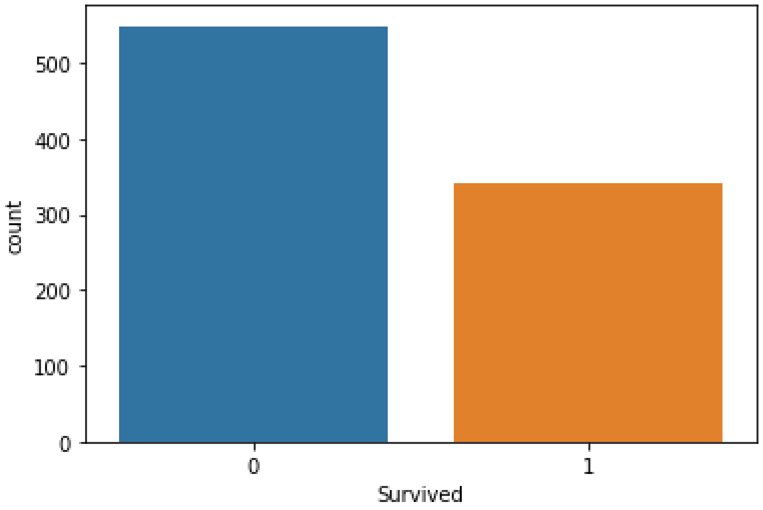
In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
titanic_data = pd.read_csv('../input/titanicdataset-traincsv/train.csv')
sb.countplot(x = 'Survived',data=titanic_data)
titanic_data
```

Out[1]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 886 | 887 | 0 | 2 | Montvila, Rev. Juozas | male | 27.0 | 0 | 0 | 211536 |
| 887 | 888 | 1 | 1 | Graham, Miss. Margaret Edith | female | 19.0 | 0 | 0 | 112053 |
| 888 | 889 | 0 | 3 | Johnston, Miss. Catherine Helen "Carrie" | female | NaN | 1 | 2 | W./C. 6607 |
| 889 | 890 | 1 | 1 | Behr, Mr. Karl Howell | male | 26.0 | 0 | 0 | 111369 |
| 890 | 891 | 0 | 3 | Dooley, Mr. Patrick | male | 32.0 | 0 | 0 | 370376 |

891 rows × 12 columns



*PART_02*

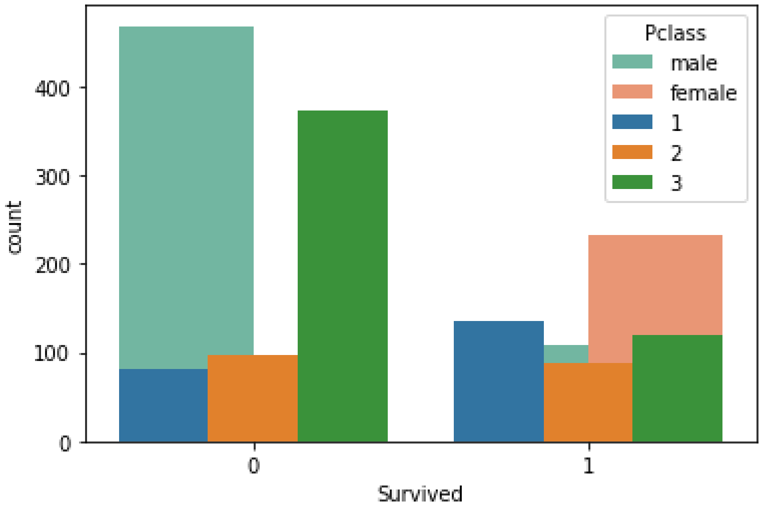
**Also plot survived comparison by class and by gender**

In [2]:

```
import pandas as pd
import seaborn as sb
titanic_data = pd.read_csv('../input/titanicdataset-traincsv/train.csv')
sb.countplot(x='Survived',data=titanic_data,palette = "Set2",hue='Sex')
sb.countplot(x='Survived',data=titanic_data,hue='Pclass')
```

Out[2]:

```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



*PART_03*

**Use the “groupby” function of Pandas to group the mean values (of all features) of passengers” gender and class.**

In [3]:

```
import pandas as pd
import seaborn as sb
titanic_data = pd.read_csv('../input/titanicdataset-traincsv/train.csv')
titanic_data.groupby('Sex').mean(),titanic_data.groupby('Pclass').mean()
```

Out[3]:

```
(        PassengerId  Survived    Pclass        Age     SibSp    
Parch  \
 Sex                                                              
 female   431.028662  0.742038  2.159236  27.915709  0.694268  0.
649682   
 male     454.147314  0.188908  2.389948  30.726645  0.429809  0.
235702   


              Fare  
 Sex                
 female  44.479818  
 male    25.523893  ,
         PassengerId  Survived        Age     SibSp     Parch    
Fare
 Pclass                                                          
 1         461.597222  0.629630  38.233441  0.416667  0.356481  8
4.154687
 2         445.956522  0.472826  29.877630  0.402174  0.380435  2
0.662183
 3         439.154786  0.242363  25.140620  0.615071  0.393075  1
3.675550)
```

**Explore how to add and delete column from a data frame, explain and illustrate with an example. Also define concept of axis in pandas.**

Answer:

- Using DataFrame.insert() to add a column

> df.insert(2, "Age", [21, 23, 24, 21], True),"2" is the position of the column, then column name and its data and set the inplace argument is True. If we change the inplace argument to False then we can't rename it.

- Remove column name 'A', we use the drop() method. i.e. df.drop(['A'], axis = 1)
- The axis argument is either 0 when it indicates rows and 1 when it is used to drop columns.

**Generate a random data frame of 20 rows and 04 columns and change their columns names.**

In [4]:

```
import pandas as pd
import numpy as np
data = pd.DataFrame(np.random.rand(20,4))
data.columns = ['A','B','C','D']
data
```

Out[4]:

| | A | B | C | D |
|---|---|---|---|---|
| 0 | 0.069831 | 0.447289 | 0.025863 | 0.828246 |
| 1 | 0.084170 | 0.138675 | 0.323775 | 0.118306 |
| 2 | 0.714977 | 0.575612 | 0.948237 | 0.258701 |
| 3 | 0.997996 | 0.073606 | 0.776104 | 0.289549 |
| 4 | 0.663651 | 0.305138 | 0.950782 | 0.321370 |
| 5 | 0.545764 | 0.397225 | 0.824474 | 0.743121 |
| 6 | 0.172301 | 0.025188 | 0.229278 | 0.505008 |
| 7 | 0.781613 | 0.081253 | 0.981055 | 0.788419 |
| 8 | 0.012391 | 0.865172 | 0.426999 | 0.083712 |
| 9 | 0.603591 | 0.666125 | 0.991409 | 0.967289 |
| 10 | 0.905258 | 0.681688 | 0.680799 | 0.826976 |
| 11 | 0.717693 | 0.828127 | 0.077290 | 0.263329 |
| 12 | 0.241906 | 0.470108 | 0.515302 | 0.370586 |
| 13 | 0.598966 | 0.180303 | 0.856625 | 0.885775 |
| 14 | 0.952100 | 0.646428 | 0.972059 | 0.909470 |
| 15 | 0.481727 | 0.703221 | 0.070683 | 0.968710 |
| 16 | 0.431279 | 0.713759 | 0.678772 | 0.125264 |
| 17 | 0.916120 | 0.701345 | 0.578164 | 0.437985 |
| 18 | 0.713053 | 0.775857 | 0.525952 | 0.070901 |
| 19 | 0.675262 | 0.323636 | 0.788212 | 0.560643 |

### Apply logical conditions to print values greater than 0.5 in any one column.

In [5]:

```
import pandas as pd
import numpy as np
Data = pd.DataFrame(np.random.rand(20,4))
Data = Data[Data > 0.5]
Data
```

Out[5]:

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | NaN | 0.681447 | 0.523002 | NaN |
| 1 | NaN | 0.915859 | NaN | NaN |
| 2 | 0.925745 | NaN | NaN | NaN |
| 3 | NaN | 0.689410 | 0.712809 | 0.791262 |
| 4 | 0.806964 | 0.905816 | 0.695568 | NaN |
| 5 | NaN | NaN | NaN | NaN |
| 6 | 0.610038 | NaN | NaN | NaN |
| 7 | 0.887043 | 0.840946 | NaN | NaN |
| 8 | NaN | NaN | 0.541174 | 0.798168 |
| 9 | 0.594403 | 0.660719 | 0.850070 | 0.819968 |
| 10 | 0.976767 | NaN | NaN | 0.696168 |
| 11 | 0.828213 | 0.657629 | 0.716942 | 0.724564 |
| 12 | 0.717707 | 0.532712 | 0.804777 | 0.908158 |
| 13 | NaN | 0.894761 | NaN | 0.948371 |
| 14 | 0.549902 | 0.679874 | NaN | NaN |
| 15 | NaN | 0.871706 | NaN | NaN |
| 16 | 0.548290 | NaN | NaN | 0.597293 |
| 17 | NaN | 0.828756 | NaN | 0.760227 |
| 18 | NaN | 0.864193 | 0.842092 | NaN |
| 19 | NaN | NaN | NaN | 0.754899 |

**Explore sort function in Pandas, apply ascending and descending sorting according.**

In [6]:

```
import pandas as pd
import numpy as np
data = pd.DataFrame(np.random.rand(20,4))
data.columns = ['A','B','C','D']
#data of column A is sort in ascending Order and Column B sort in descending order.
data.sort_values(by='A', ascending=True),data.sort_values(by='B',ascending=False)
```

```
Out[6]:

(          A         B         C         D
 17  0.031138  0.517919  0.793281  0.657622
 18  0.102630  0.369786  0.731633  0.700404
 0   0.118831  0.491025  0.738283  0.969084
 10  0.146729  0.120487  0.269104  0.383900
 16  0.176165  0.751997  0.190086  0.940886
 14  0.205057  0.415023  0.379708  0.816027
 5   0.217516  0.835165  0.071027  0.608958
 3   0.235302  0.853471  0.053725  0.622969
 9   0.390502  0.941143  0.852058  0.032018
 1   0.408124  0.211590  0.735392  0.585510
 8   0.492108  0.919017  0.553552  0.212700
 4   0.500963  0.060900  0.573474  0.342855
 15  0.668436  0.783976  0.158918  0.137596
 11  0.702004  0.991834  0.378603  0.625031
 7   0.748709  0.251168  0.937220  0.589441
 6   0.783395  0.095850  0.248631  0.224988
 19  0.800210  0.343622  0.928233  0.197981
 2   0.825349  0.319851  0.851632  0.945442
 13  0.898191  0.942837  0.503853  0.443568
 12  0.968203  0.139958  0.533965  0.808591,
           A         B         C         D
 11  0.702004  0.991834  0.378603  0.625031
 13  0.898191  0.942837  0.503853  0.443568
 9   0.390502  0.941143  0.852058  0.032018
 8   0.492108  0.919017  0.553552  0.212700
 3   0.235302  0.853471  0.053725  0.622969
 5   0.217516  0.835165  0.071027  0.608958
 15  0.668436  0.783976  0.158918  0.137596
 16  0.176165  0.751997  0.190086  0.940886
 17  0.031138  0.517919  0.793281  0.657622
 0   0.118831  0.491025  0.738283  0.969084
 14  0.205057  0.415023  0.379708  0.816027
 18  0.102630  0.369786  0.731633  0.700404
 19  0.800210  0.343622  0.928233  0.197981
 2   0.825349  0.319851  0.851632  0.945442
 7   0.748709  0.251168  0.937220  0.589441
 1   0.408124  0.211590  0.735392  0.585510
 12  0.968203  0.139958  0.533965  0.808591
 10  0.146729  0.120487  0.269104  0.383900
 6   0.783395  0.095850  0.248631  0.224988
 4   0.500963  0.060900  0.573474  0.342855)
```

**Loc and iloc are two functions of pandas for accessing the data from the data frame. Define the difference between them and use them each in at least 2 example**

- loc is label-based, which means that we have to specify the name of the rows and columns that we need to filter out.
- On the other hand, iloc is integer index-based. So here, we have to specify rows and columns by their integer index.

> loc[row_label, column_label]

> iloc[row_position, column_position]

**Example 01: LOC[]**

In [7]:

```
import pandas as pd
Data = pd.DataFrame({'A':[1,23,4,12],'B':[34,14,62,14],'C':["Asif","Sarwar",2,31],'D':[2,44,52,1.2]})
index = ['A','B','C','D']
Data.loc[:,'A']
```

Out[7]:

```
0     1
1    23
2     4
3    12
Name: A, dtype: int64
```

**Example 02: LOC[]**

In [8]:

```
import pandas as pd
data = pd.read_csv('../input/world-happiness/2019.csv')
data.loc[1,'Score'],data
```

```
Out[8]:

(7.6,
      Overall rank         Country or region  Score  GDP per capita
\
 0               1                   Finland  7.769           1.340
 1               2                   Denmark  7.600           1.383
 2               3                    Norway  7.554           1.488
 3               4                   Iceland  7.494           1.380
 4               5               Netherlands  7.488           1.396
 ..            ...                       ...    ...             ...
 151           152                    Rwanda  3.334           0.359
 152           153                  Tanzania  3.231           0.476
 153           154               Afghanistan  3.203           0.350
 154           155  Central African Republic  3.083           0.026
 155           156               South Sudan  2.853           0.306

      Social support  Healthy life expectancy  Freedom to make life
choices  \
 0             1.587                    0.986
0.596
 1             1.573                    0.996
0.592
 2             1.582                    1.028
0.603
 3             1.624                    1.026
0.591
 4             1.522                    0.999
0.557
 ..              ...                      ...
...
 151           0.711                    0.614
0.555
 152           0.885                    0.499
0.417
 153           0.517                    0.361
0.000
 154           0.000                    0.105
0.225
 155           0.575                    0.295
0.010

      Generosity  Perceptions of corruption
 0         0.153                      0.393
 1         0.252                      0.410
 2          0.271                      0.341  
 3          0.354                      0.118  
 4          0.322                      0.298  
 ..           ...                        ...  
 151        0.217                      0.411  
 152        0.276                      0.147  
 153        0.158                      0.025  
 154        0.235                      0.035  
 155        0.202                      0.091  

 [156 rows x 9 columns])
```

### Example 01: iloc[]

In [9]:

```
import pandas as pd
import numpy as np
Data = pd.read_csv('../input/world-happiness/2019.csv')
Data.iloc[:,1]
```

Out[9]:

```
0                       Finland
1                       Denmark
2                        Norway
3                       Iceland
4                   Netherlands
                 ...           
151                      Rwanda
152                    Tanzania
153                 Afghanistan
154    Central African Republic
155                 South Sudan
Name: Country or region, Length: 156, dtype: object
```

### Example 02: iloc[]

In [10]:

```
import pandas as pd
Data = pd.read_csv('../input/world-happiness/2019.csv')
Data.iloc[1,2],Data
```

```
Out[10]:

(7.6,
      Overall rank        Country or region  Score  GDP per capita  
\
 0               1                  Finland  7.769           1.340   
 1               2                  Denmark  7.600           1.383   
 2               3                   Norway  7.554           1.488   
 3               4                  Iceland  7.494           1.380   
 4               5              Netherlands  7.488           1.396   
 ..            ...                      ...    ...             ...   
 151           152                   Rwanda  3.334           0.359   
 152           153                 Tanzania  3.231           0.476   
 153           154              Afghanistan  3.203           0.350   
 154           155 Central African Republic  3.083           0.026   
 155           156              South Sudan  2.853           0.306   

      Social support  Healthy life expectancy  Freedom to make life 
choices  \
 0             1.587                    0.986                        
0.596   
 1             1.573                    0.996                        
0.592   
 2             1.582                    1.028                        
0.603   
 3             1.624                    1.026                        
0.591   
 4             1.522                    0.999                        
0.557   
 ..              ...                      ...                        
...   
 151           0.711                    0.614                        
0.555   
 152           0.885                    0.499                        
0.417   
 153           0.517                    0.361                        
0.000   
 154           0.000                    0.105                        
0.225   
 155           0.575                    0.295                        
0.010   

      Generosity  Perceptions of corruption  
 0         0.153                      0.393  
 1         0.252                      0.410  
 2         0.271                       0.341
 3         0.354                       0.118
 4         0.322                       0.298
 ..          ...                         ...
 151       0.217                       0.411
 152       0.276                       0.147
 153       0.158                       0.025
 154       0.235                       0.035
 155       0.202                       0.091

 [156 rows x 9 columns])
```