In [3]:

```
# Pandas Unique and Count Functions
# Here, you will see how you can get a list of unique
# values, the number of all unique values, and records per
# unique value from a column in a Pandas dataframe.
# You will be using the Titanic dataset once again.

import matplotlib.pyplot as plt
import seaborn as sns
# sets the default style for plotting
sns.set_style("darkgrid")
titanic_data = sns.load_dataset('titanic')
titanic_data.head()
```

Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

In [4]:

```
# To find the number of all the unique values in a column, you
# can use the unique() function. The script below returns all the
# unique values from the class column from the Titanic dataset.

titanic_data["class"].unique()
```

Out[4]:

```
['Third', 'First', 'Second']
Categories (3, object): ['Third', 'First', 'Second']
```

In [5]:

```
# To get the count of unique values, you can use the nunique()
# method, as shown in the script below.

titanic_data["class"].nunique()
```

Out[5]:

```
3
```

In [6]:

```
# To get the count of non-null values for all the columns in your
# dataset, you may call the count() method on the Pandas
# dataframe. The following script prints the count of the total
# number of non-null values in all the columns of the Titanic
# dataset.

titanic_data.count()
```

Out[6]:

```
survived       891
pclass         891
sex            891
age            714
sibsp          891
parch          891
fare           891
embarked       889
class          891
who            891
adult_male     891
deck           203
embark_town    889
alive          891
alone          891
dtype: int64
```

In [7]:

```
# Finally, if you want to find the number of records for all the
# unique values in a dataframe column, you may use the
# value_counts() function.
# The script below returns counts of records for all the unique
# values in the class column.

titanic_data["class"].value_counts()
```

Out[7]:

```
Third     491
First     216
Second    184
Name: class, dtype: int64
```

In [8]:

```
# Grouping Data with GroupBy
# To group data by a column value, you can use the groupby()
# function of the Pandas dataframe. You need to pass the
# column as the parameter value to the groupby() function.
# The script below groups the data in the Titanic dataset by the
# class column. Next, the type of object returned by the
# groupby() function is also printed.

titanic_gbclass = titanic_data.groupby("class")
type(titanic_gbclass)
```

Out[8]:

```
pandas.core.groupby.generic.DataFrameGroupBy
```

In [9]:

```
# The above output shows that the groupby() function returns
# the DataFrameGroupBy object. You can use various attributes
# and functions of this object to get various information about
# different groups.
# For instance, to see the number of groups, you can use the
# ngroups attribute, which returns the number of groups
# (unique values). Since the number of groups in the class
# columns is 3, you will see 3 printed in the output of the script.

titanic_gbclass.ngroups
```

Out[9]:

```
3
```

In [10]:

```
# You can use the size() function to get the number of records
# in each group.

titanic_gbclass.size()
```

Out[10]:

```
class
First     216
Second    184
Third     491
dtype: int64
```

In [11]:

```
# Finally, you can also get the row indexes for records in a
# particular group. The script below returns the row indexes for
# rows where the class column contains “First.”

titanic_gbclass.groups["First"]
```

Out[11]:

```
Int64Index([  1,   3,   6,  11,  23,  27,  30,  31,  34,  35,
            ...
            853, 856, 857, 862, 867, 871, 872, 879, 887, 889],
           dtype='int64', length=216)
```

In [12]:

```
# You can also get the first or last record from each group using
# the first() and last() functions, respectively.
# As an example, the following script returns the last record
# from each group in the class column.

titanic_gbclass.last()
```

Out[12]:

class	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	adult_male	deck	(
First	1	1	male	26.0	0	0	30.00	C	man	True	C	
Second	0	2	male	27.0	0	0	13.00	S	man	True	E	
Third	0	3	male	32.0	0	0	7.75	Q	man	True	E	

In [13]:

```
# You can also get a dataframe that contains records belonging
# to a subgroup using the get_group() function. The following
# script returns all records from the group named “Second”
# from the class column.

titanic_second_class = titanic_gbclass.get_group("Second")
titanic_second_class.head()
```

Out[13]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	adult_male	deck
9	1	2	female	14.0	1	0	30.0708	C	child	False	NaN
15	1	2	female	55.0	0	0	16.0000	S	woman	False	NaN
17	1	2	male	NaN	0	0	13.0000	S	man	True	NaN
20	0	2	male	35.0	0	0	26.0000	S	man	True	NaN
21	1	2	male	34.0	0	0	13.0000	S	man	True	D

In [14]:

```
# The DataFrameByGroup object can also be used to perform
# aggregate functions. For instance, you can get the maximum
# age in all the groups in the class column using the max()
# function, as shown in the following script.

titanic_gbclass.age.max()
```

Out[14]:

```
class
First     80.0
Second    70.0
Third     74.0
Name: age, dtype: float64
```

In [15]:

```
# Similarly, you can also get information based on various
# aggregate functions bypassing the list of functions to the
# agg() method.
# For instance, the script below returns the maximum, minimum,
# median, and mean, and count of age values in different groups
# in the class column.

titanic_gbclass.fare.agg(['max', 'min', 'count', 'median', 'mean'])
```

Out[15]:

class	max	min	count	median	mean
First	512.3292	0.0	216	60.2875	84.154687
Second	73.5000	0.0	184	14.2500	20.662183
Third	69.5500	0.0	491	8.0500	13.675550