Experiment No.: 4

Title: Write a program to visualize data using data visualization library seaborn.
Objectives:

1. To learn how to visualize data using visualization library Seaborn.

Theory:

Data visualization is viewed by many disciplines as a modern equivalent of visualcommunication. It involves the creation and study of the visual representation of data.To communicate information clearly and efficiently, data visualization uses statistical graphics, plots, information graphics and other tools. It makes complex data more accessible, understandable and usable.Data visualization is both an art and a science.

Seaborn has many of its own high-level plotting routines, but it can also overwrite Matplotlib's default parameters and in turn get even simple Matplotlib scripts to produce vastly superior output. We can set the style by calling Seaborn's**set()** method. By convention, Seaborn is imported as **sns**:

```
importseabornassns
sns.set()
# same plotting code as in Matplotlib can produce better visualization!
# Create some data
rng=np.random.RandomState(0)
x =np.linspace(0, 10, 500)
y =np.cumsum(rng.randn(500, 6), 0)
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```

The main idea of Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration, and even some statistical model fitting.

Visualizing statistical relationships:

Seaborn provides various functions out of which one we will use most is **relplot()**. This is a figure-level function for visualizing statistical relationships using two common approaches: **scatter plots and line plots**.

A function called **relplot()** combines a **FacetGrid** (means that it is easy to add faceting variables to visualize higher-dimensional relationships) with one of two axes-level functions:

- scatterplot() (with kind="**scatter**"; the default)
- lineplot() (with kind="**line**")

These functions can be quite illuminating because they use simple and easily-understood representations of data that can nevertheless represent complex dataset structures. They can do so because they plot two-dimensional graphics that can be enhanced by mapping up to three additional variables using the semantics of hue, size, and style.

```
tips=sns.load_dataset("tips")

sns.relplot(x="total_bill", y="tip", data=tips);
```

While the points are plotted in two dimensions, another dimension can be added to the plot by coloring the points according to a third variable. In seaborn, this is referred to as using a “hue semantic”, because the color of the point gains meaning:

```
sns.relplot(x="total_bill", y="tip", hue="smoker", data=tips);
```

To emphasize the difference between the classes, and to improve accessibility, you can use a different marker stylesand other parameters for each class:

```
sns.relplot(x="total_bill", y="tip", hue="smoker", style="smoker",data=tips);
sns.relplot(x="total_bill", y="tip", hue="smoker", style="time", data=tips);
sns.relplot(x="total_bill", y="tip", hue="size", data=tips);
sns.relplot(x="total_bill", y="tip", hue="size", palette="ch:r=-.5,l=.75", data=tips);
```

Plotting Categorical Data

```
importseabornassns
importmatplotlib.pyplotasplt
sns.set(style="ticks", color_codes=True)
tips=sns.load_dataset("tips")
1. sns.catplot(x="day", y="total_bill", data=tips);
2. sns.catplot(x="day", y="total_bill", jitter=False, data=tips);
3. sns.catplot(x="day", y="total_bill", kind="swarm", data=tips);
4. sns.catplot(x="day", y="total_bill", hue="sex", kind="swarm", data=tips);
5. sns.catplot(x="total_bill", y="day", hue="time", kind="swarm", data=tips);
```

Boxplots:

This uses boxplot() function. Box plot shows the three quartile values of the distribution along with extreme values. Each value in the boxplot corresponds to an actual observation in the data.

```
1. sns.catplot(x="day", y="total_bill", kind="box", data=tips);
2. sns.catplot(x="day", y="total_bill", hue="smoker", kind="box", data=tips);
3. diamonds=sns.load_dataset("diamonds")
sns.catplot(x="color", y="price", kind="boxen",data=diamonds.sort_values("color"));
```

Bar plots:

In seaborn, the barplot() function operates on a full dataset and applies a function to obtain the estimate (taking the mean by default). When there are multiple observations in each category, it also uses bootstrapping to compute a confidence interval around the estimate and plots that using error bars:

```
1. titanic=sns.load_dataset("titanic")

sns.catplot(x="sex", y="survived", hue="class", kind="bar", data=titanic);

2. sns.catplot(x="deck", kind="count", palette="ch:.25", data=titanic);

3. sns.catplot(y="deck", hue="class", kind="count",palette="pastel", edgecolor=".6",

data=titanic);
```

Point plots:

Thisplots use pointplot() function. It also encodes the value of the estimate with height on the other axis, but rather than showing a full bar, it plots the point estimate and confidence interval. Additionally, pointplot() connects points from the same hue category. This makes it easy to see how the main relationship is changing as a function of the hue semantic, because your eyes are quite good at picking up on differences of slopes:

```
1. sns.catplot(x="sex", y="survived", hue="class", kind="point", data=titanic);

2. sns.catplot(x="class", y="survived", hue="sex",palette={"male": "g", "female": "m"},

markers=["^", "o"], linestyles=["-", "--"],kind="point", data=titanic);
```

Showing multiple relationships with facets:

Just like relplot(),catplot() is built on a FacetGrid means that it is easy to add faceting variables to visualize higher-dimensional relationships:

```
1. sns.catplot(x="day", y="total_bill", hue="smoker",col="time", aspect=.6,kind="swarm",
data=tips);

2. g=sns.catplot(x="fare", y="survived", row="class",kind="box", orient="h",height=1.5,
aspect=4,data=titanic.query("fare > 0"))

g.set(xscale="log");
```

Histograms&Kernel density estimation:

Rather than a histogram, we can get a smooth estimate of the distribution using a kernel density estimation, which Seaborn does with **sns.kdeplot()**.The kernel density estimate may be less familiar, but it can be a useful tool for plotting the shape of a distribution. Like the histogram, the KDE plots encode the density of observations on one axis with height along the other axis:

```
data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)

data = pd.DataFrame(data, columns=['x', 'y'])


1. HISTOGRAM:     for col in 'xy':

                  plt.hist(data[col], normed=True, alpha=0.5)

2. KDEPLOT:       for col in 'xy':

                  sns.kdeplot(data[col], shade=True)

3. Histograms and KDE can be combined using distplot:

                  sns.distplot(data['x'])

                  sns.distplot(data['y']);

4. If we pass the full two-dimensional dataset to kdeplot, we will get a two-dimensional
visualization of the data:

                  sns.kdeplot(data);
```

Violinplots:

A different approach is a violinplot(), which combines a boxplot with the kernel density estimation procedure.The downside is that, because the violinplot uses a kernel density estimate (KDE), there are some other parameters that may need tweaking, adding some complexity relative to the straightforward boxplot:

```
1.sns.catplot(x="total_bill", y="day", hue="time", kind="violin", data=tips);
2.sns.catplot(x="total_bill",y="day", hue="time",kind="violin",bw=.15, cut=0,data=tips)
```

Visualizing pairwise relationships in a dataset:

To plot multiple pairwise bivariate distributions in a dataset, you can use the pairplot() function.It is useful for exploring correlations between multidimensional data, when you'd like to plot all pairs of values against each other.

```
iris=sns.load_dataset("iris")

iris.head()

sns.pairplot(iris, hue='species', size=2.5);
```

Key concepts: Data Visualization