DIFFERENTIAL PRIVACY IMPACT BASED ON VARYING PRIVACY BUDGET

Given a histogram (for a data set with n=100 observations) having 10 bins, and the following counts:

[4, 9, 20, 15, 17, 19, 8, 5, 2, 1)

Explore a range of values for epsilon (a measure of the privacy budget) from very small (0.01) to rather large (10), in the following steps:

[0.01, 0.05, 0.1, 0.5, 1, 5, 10]

Using the Laplace method (and either R or Python), produce a set of 7 altered histogram counts, with varying amounts of noise, for each choice of epsilon.

From among the 7 options, select one that you would choose to release. Justify your explanation, considering the balance between accuracy and privacy.

In [17]:
```
# -----------------------------------------------------------------------------------
--
# Initial Configuration
# -----------------------------------------------------------------------------------
--

import random
import pandas as pd
import statistics as st
import numpy as np
import pprint as pp
import math as mth
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy
#import pyspark
import os
import xlrd
from scipy.stats import laplace
```

In [18]:

```
#
--------------------------------------------------------------------------------
--
# Generating bincounts based on epsilon set
#
--------------------------------------------------------------------------------
--

# Defining Observation Values
vals = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# Defining Initia Bin Count
bin_count = [4, 9, 20, 15, 17, 19, 8, 5, 2, 1]

# Defining Epsilon Factors
factors = [0.01, 0.05, 0.1, 0.5, 1.0, 5.0, 10.0]

#
--------------------------------------------------------------------------------
--
# Configuring a list of lists to store all bin counts (8) including the initial on
e.
#
--------------------------------------------------------------------------------
--

# Adding the initial list to list of bin count lists
reps = list()
reps.append(bin_count)

# Genrating all 7 bin count lists
for i,eps in enumerate(factors):
      reps.append([j + abs(int(round(float((np.random.laplace(j, 1/eps, 1))),0))) f
or j in bin_count])
```

In [19]:

```
#
--------------------------------------------------------------------------------
--
# Examining generated bin counts (Skip first list as it is default)
#
--------------------------------------------------------------------------------
--
reps
```

Out[19]:

```
[[4, 9, 20, 15, 17, 19, 8, 5, 2, 1],
 [94, 236, 65, 138, 22, 415, 23, 64, 45, 36],
 [100, 28, 32, 37, 68, 50, 31, 17, 4, 68],
 [12, 34, 28, 26, 35, 39, 28, 10, 4, 2],
 [8, 21, 43, 30, 36, 37, 14, 7, 5, 1],
 [8, 18, 40, 30, 34, 38, 16, 13, 4, 2],
 [8, 18, 41, 29, 34, 38, 16, 10, 4, 2],
 [8, 18, 40, 30, 34, 38, 16, 10, 4, 2]]
```

In [20]:
```
# ---------------------------------------------------------------------------------
--
# Scaling all bin lists to 100 for apple to apple comparison of epsilon factor impa
ct
# ---------------------------------------------------------------------------------
--
all_bins = reps

bins_new = list()
for bin_x in all_bins:
    x = 100/sum(bin_x)
    y = [int(round(i*x,0)) for i in bin_x]
    bins_new.append(y)

reps = bins_new
reps
```

Out[20]:
```
[[4, 9, 20, 15, 17, 19, 8, 5, 2, 1],
 [8, 21, 6, 12, 2, 36, 2, 6, 4, 3],
 [23, 6, 7, 9, 16, 11, 7, 4, 1, 16],
 [6, 16, 13, 12, 16, 18, 13, 5, 2, 1],
 [4, 10, 21, 15, 18, 18, 7, 3, 2, 0],
 [4, 9, 20, 15, 17, 19, 8, 6, 2, 1],
 [4, 9, 20, 14, 17, 19, 8, 5, 2, 1],
 [4, 9, 20, 15, 17, 19, 8, 5, 2, 1]]
```

In [21]:
```
# ---------------------------------------------------------------------------------
-------
# Converting the Bin Counts (Initial and DPed ones) into a directory
# Set0 = Initial Bin Count
# ---------------------------------------------------------------------------------
-------
bins_dir = {}

for i, rep in enumerate(reps):
    key_value = 'Set' + str(i)
    bins_dir[key_value] = rep

bins_dir
```

Out[21]:
```
{'Set0': [4, 9, 20, 15, 17, 19, 8, 5, 2, 1],
 'Set1': [8, 21, 6, 12, 2, 36, 2, 6, 4, 3],
 'Set2': [23, 6, 7, 9, 16, 11, 7, 4, 1, 16],
 'Set3': [6, 16, 13, 12, 16, 18, 13, 5, 2, 1],
 'Set4': [4, 10, 21, 15, 18, 18, 7, 3, 2, 0],
 'Set5': [4, 9, 20, 15, 17, 19, 8, 6, 2, 1],
 'Set6': [4, 9, 20, 14, 17, 19, 8, 5, 2, 1],
 'Set7': [4, 9, 20, 15, 17, 19, 8, 5, 2, 1]}
```

In [22]:
```
#
--------------------------------------------------------------------------------------
-------------
# Generating the observations based on bin counts for each epsilon value based intr
oduced error
# Set0 = Initial Bin Count
#
--------------------------------------------------------------------------------------
-------------
results = {}

#Looping through all bin sets (8)
for i, rep in enumerate(reps):
    key_value = 'Readings' + str(i)
    value_set = list()

    #Looping through all values for each bin set to generate the actual observation
values as per bin count
    for j, val in enumerate(vals):
        temp_list = np.ndarray.tolist(np.repeat(vals[j], rep[j]))
        for v1 in temp_list:
            value_set.append(v1)
    results[key_value] = value_set

str(results)
```

Out[22]:


```
"{'Readings0': ['a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b',
'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c',
'c', 'c', 'c', 'c', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd',
'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e',
'e', 'e', 'e', 'e', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'h',
'h', 'h', 'h', 'h', 'i', 'i', 'j'], 'Readings1': ['a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b',
'b', 'b', 'b', 'b', 'b', 'b', 'b', 'c', 'c', 'c', 'c', 'c', 'c', 'd', 'd', 'd',
'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'e', 'e', 'f', 'f', 'f', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'g',
'g', 'h', 'h', 'h', 'h', 'h', 'h', 'i', 'i', 'i', 'i', 'j', 'j', 'j'], 'Readings
2': ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b', 'c', 'c',
'c', 'c', 'c', 'c', 'c', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'e', 'e',
'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'g', 'g', 'g', 'g', 'g', 'g', 'g',
'h', 'h', 'h', 'h', 'i', 'j', 'j', 'j', 'j', 'j', 'j', 'j', 'j', 'j', 'j', 'j',
'j', 'j', 'j', 'j', 'j'], 'Readings3': ['a', 'a', 'a', 'a', 'a', 'a', 'b', 'b',
'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'c', 'c',
'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'd', 'd', 'd', 'd', 'd',
'd', 'd', 'd', 'd', 'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e',
'e', 'e', 'e', 'e', 'e', 'e', 'e', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'g', 'g', 'g', 'g', 'g', 'g', 'g',
'g', 'g', 'g', 'g', 'g', 'g', 'h', 'h', 'h', 'h', 'h', 'i', 'i', 'j'], 'Readings
4': ['a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'c',
'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c',
'c', 'c', 'c', 'c', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd',
'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e',
'e', 'e', 'e', 'e', 'e', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'h', 'h',
'h', 'i', 'i'], 'Readings5': ['a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b',
'b', 'b', 'b', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c',
'c', 'c', 'c', 'c', 'c', 'c', 'c', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd',
'd', 'd', 'd', 'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e',
'e', 'e', 'e', 'e', 'e', 'e', 'e', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'g', 'g', 'g', 'g', 'g', 'g',
'g', 'g', 'h', 'h', 'h', 'h', 'h', 'h', 'i', 'i', 'j'], 'Readings6': ['a', 'a',
'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'c', 'c', 'c', 'c', 'c',
'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'd',
'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'e', 'e', 'e',
'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f',
'f', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'h', 'h', 'h', 'h', 'h', 'i', 'i',
'j'], 'Readings7': ['a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b',
'b', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c', 'c',
'c', 'c', 'c', 'c', 'c', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd', 'd',
'd', 'd', 'd', 'd', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e', 'e',
'e', 'e', 'e', 'e', 'e', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f',
'f', 'f', 'f', 'f', 'f', 'f', 'f', 'f', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g',
'h', 'h', 'h', 'h', 'h', 'i', 'i', 'j']}"
```

In [23]:
```
# ----------------------------------------------------
# Creating a Custom Palette
# ----------------------------------------------------
custom_palette =   ['#78C850',  # Grass
                    '#F08030',  # Fire
                    '#6890F0',  # Water
                    '#A8B820',  # Bug
                    '#A8A878',  # Normal
                    '#A040A0',  # Poison
                    '#F8D030',  # Electric
                    '#E0C068',  # Ground
                    '#EE99AC',  # Fairy
                    '#C03028',  # Fighting
                    '#F85888',  # Psychic
                    '#B8A038',  # Rock
                    '#705898',  # Ghost
                    '#98D8D8',  # Ice
                    '#7038F8',  # Dragon
                   ]
```

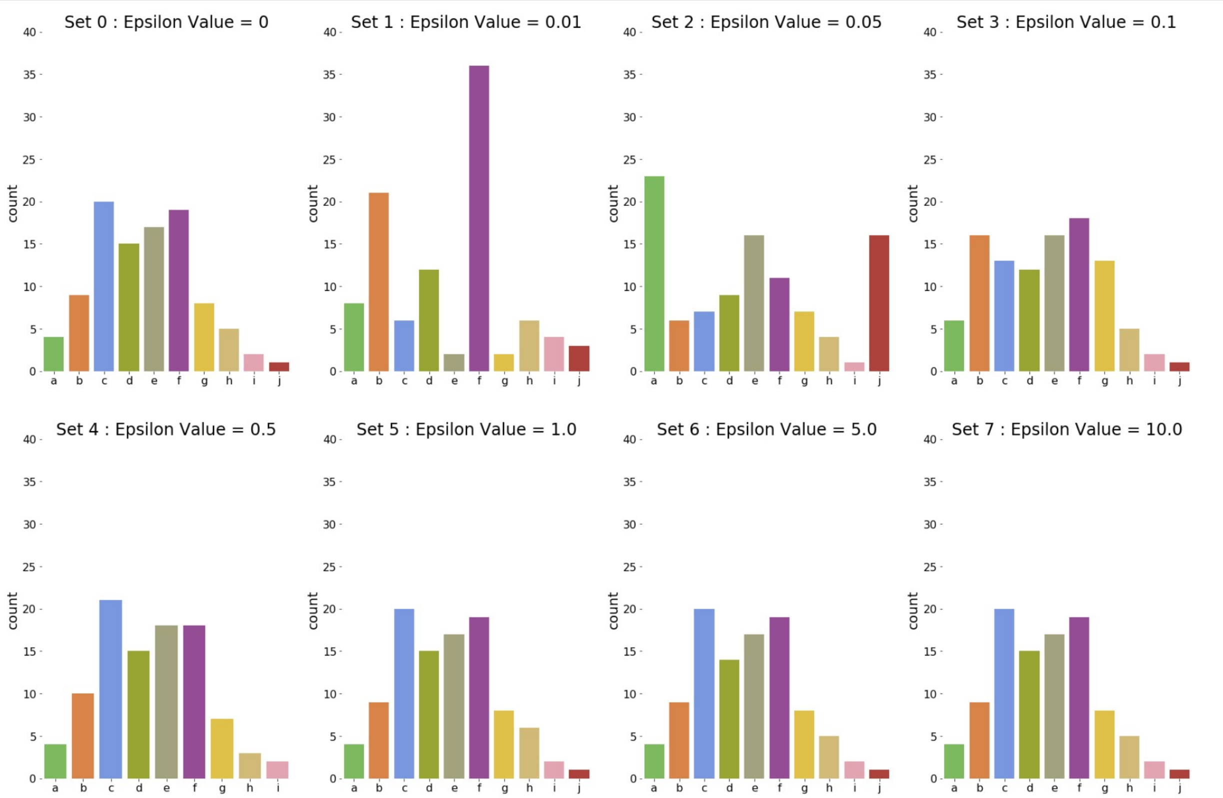
In [24]:
```
# Setting up the Plot Diagram (2 * 4)
sns.color_palette(custom_palette)
fig, axes = plt.subplots(2, 4, figsize = (30,20))

rows = axes.shape[0]
cols = axes.shape[1]
id = 0
# Configuring and painting individual plot for each Epsilon Value on Bin Counts
for i in range(rows):
    for j in range(cols):

        # Removing all axis lines
        axes[i,j].spines['top'].set_visible(False)
        axes[i,j].spines['right'].set_visible(False)
        axes[i,j].spines['left'].set_visible(False)
        axes[i,j].spines['bottom'].set_visible(False)
        axes[i,j].set_ylim(0, 40)
        axes[i,j].set_ylabel("Count", fontsize = 20)
        axes[i,j].tick_params(labelsize = 16)
        # Assigning Key Value, Epsilon Value, Title for each plot
        key_value = 'Readings' + str(id)

        if id == 0:
            eps = 0
        else:
            eps = factors[id - 1]
        plot_title = 'Set ' + str(id) + " : Epsilon Value = " + str(eps)

        #Generating the Seaborn CountPlot for each Epsilon Value
        sns.countplot(x = results[key_value], ax = axes[i,j], palette = custom_palette).set_title(plot_title, fontsize = 24)
        id += 1
```



Above is the comparison of histograms/count plots based on 7 epsilon factors.

Set 0 depicts the initial bin counts.

Assuming that Laplace Distribution is applied correctly and is working correctly.

Considering need for accuracy and privacy, I would pick either Set 4 OR 5 which are based on Epsilon Value = 0.5 OR 1 as it randomizes the bin counts effectively and still close to original bin counts.

In other Epsilon Factors, either the bin distributions are the same as original OR they are too random to provide any value.