**Bar Charts**

**Lesson 5, video 4**

**First import these libraries**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sb**

**%matplotlib inline**

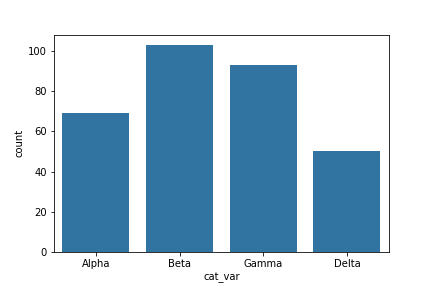
* **For import bar chart**

sb.countplot(data = df, x = 'cat\_var', color = base\_color

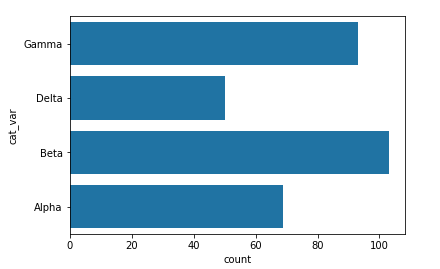
* **for color of bar chat**

base\_color = sb.color\_palette()[0]

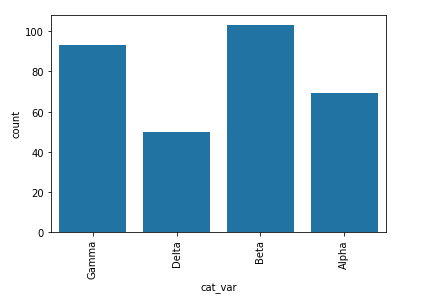
* x=’cat var’ for vertical bar



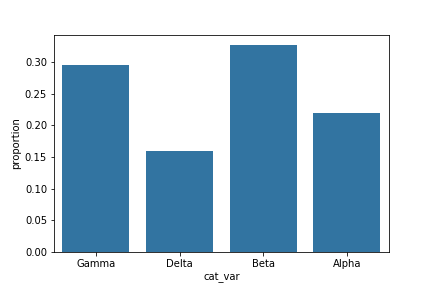
* y=’car var’ for horizontal bar



* Rotation:=
* plt.xticks(rotation = 90)
* use for horizontal the name of bar
* Alternatively, you can use matplotlib's **[xticks](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xticks.html" \t "_blank)** function and its "rotation" parameter to change the orientation in which the labels will be depicted (as degrees counter-clockwise from horizontal):

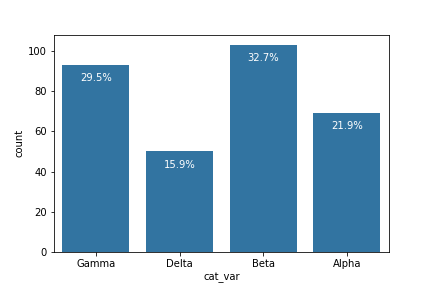


## Absolute Frequency:= actual values are use in bar chat



* **Relative Frequency:=** percentage of values are used in bar chat,

Is called **proportional of data**



## Video 9:

## Histogram:

## By default, the hist function divides the data into 10 bins, based on the range of values taken.

## In almost every case, we will want to change these settings. Usually, having only ten bins is too few to really understand the distribution of the data. And the default tick marks are often not on nice, 'round' values that make the ranges taken by each bin easy to interpret. Wouldn't it be better if I said "between 0 and 2.5" instead of "between about 0 and 2.5", and "from 2.5 to 5" instead of "from about 2.5 to 5" above?

## For this we use,

You can use descriptive statistics (e.g. via **[df['num\_var'].describe()](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.describe.html" \t "_blank)**) to gauge what minimum and maximum bin limits might be appropriate for the plot. These bin edges can be set using numpy's **[arange](https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html" \t "_blank)**function:

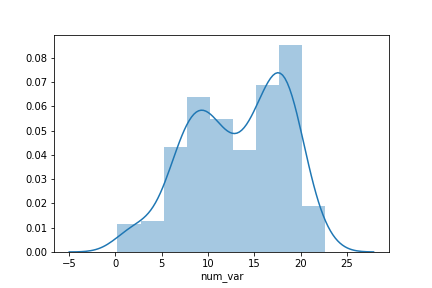
bin\_edges = np.arange(0, df['num\_var'].max()+1, 1)

plt.hist(data = df, x = 'num\_var', bins = bin\_edges)

## Alternative Approach

The seaborn function **[distplot](https://seaborn.pydata.org/generated/seaborn.distplot.html" \t "_blank)** can also be used to plot a histogram, and is integrated with other univariate plotting functions.

sb.distplot(df['num\_var'])



Note that the first argument must be the Series or array with the points to be plotted, rather than being able to specify a data source and column.

NOTE:=

sb.distplot(df['num\_var'])

here df is table name and ‘num\_var’ is column name that we want to plot

**MATPLOTLIB 2**

* Quantitative variable vs Quantitative variable use **Scatterplots**
* Quantitative variable vs qualitative variable use **Violin plots**
* qualitative variable vs qualitative variable use **Clustered bar chart**

**Video 2:**

By default, the regression function is linear, and includes a shaded confidence region for the regression estimate. In this case, since the trend looks like a log(y) ~ x relationship, plotting the regression line on the raw units is not appropriate. If we don't care about the regression line, then we could set reg\_fit = False in the regplot function call. Otherwise, if we want to plot the regression line on the observed relationship in the data, we need to transform the data, as seen in the previous lesson.

**def** **log\_trans**(x, inverse = False):

**if** **not** inverse:

**return** np.log10(x)

**else**:

**return** np.power(10, x)

sb.regplot(df['num\_var1'], df['num\_var2'].apply(log\_trans))

tick\_locs = [10, 20, 50, 100, 200, 500]

plt.yticks(log\_trans(tick\_locs), tick\_locs)

Note that the x- and y- values can be set as Series or arrays directly, instead of columns from a reference dataframe.

Overplotting:

Where a plot is created with too many overlapping points

Resolve with:

* sampling
* Transparency
* Jitter

Violin plots use to define relationship between Quantitative variable vs qualitative variable

ax = sb.countplot(data = fuel\_econ, x = 'VClass')

#ax.legend(loc = 8, ncol = 3, framealpha = 1, title = 'cat\_var2')

Sb.countplot(data=’’, x=’ ‘, hue=’ ‘)

Plt.xtricks(rotation=15);

ct\_counts = df.groupby(['cat\_var1', 'cat\_var2']).size()

ct\_counts = ct\_counts.reset\_index(name = 'count')

ct\_counts = ct\_counts.pivot(index = 'cat\_var2', columns = 'cat\_var1', values = 'count')

base\_color = sb.color\_palette()[0]