Seaborn Summary

Ahmed Mohamed Ads

Tuesday 29th April, 2025
1 Dhu al-Qi'dah 1446 AH

Seaborn

seaborn

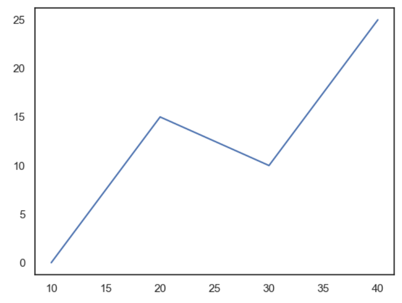
A Python library used for enhancing the style of different graphs.

Seaborn updates matplotlib's rc parameters to improve aesthetics

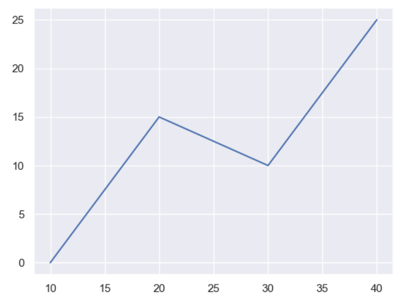
```
import seaborn as sns
sns.set()
plt.plot(x, y)
sns.set_style("white")
```

- `import seaborn as sns`: importing the seaborn library to use it with the name sns.
- `sns.set()`: function for setting the seaborn default style on the plot by updating the matplotlib's rc parameters, and see Seaborn set documentation for more details.
- `plt.plot()`: plotting the data to watch the difference.
- `sns.set_style("white")`: for making a specific background style, here a plain white background.

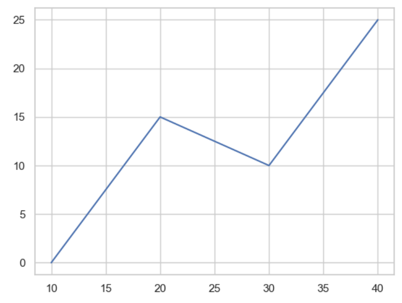
Figure 1: Diffrences between plots



(a) Original plot



(b) After using sns.set()



(c) After using sns.set_style()

Versions

```
import matplotlib            #importing matplotlib library
matplotlib.__version__       #Getting the current version of matplotlib
import seaborn as sns        #importing seaborn library as sns
sns.__version__              #Getting the current version of seaborn
```

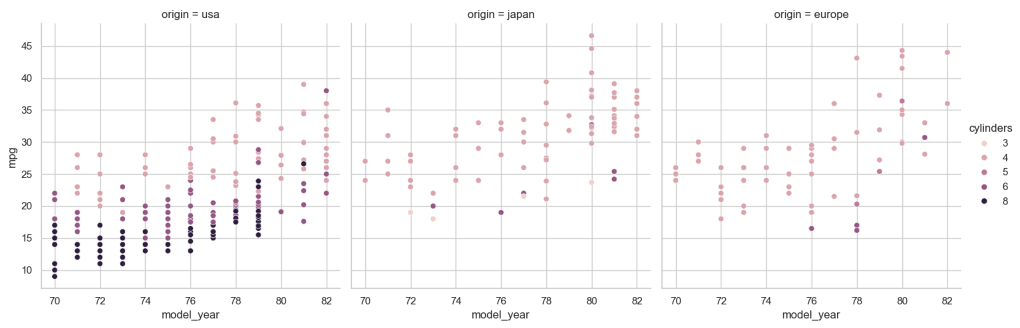
Seaborn with pandas

Let Seaborn group, aggregate, and plot your pandas dataframes

```
cars = sns.load_dataset('mpg')
type(cars)
cars.dropna(inplace=True)
cars.shape
cars.head()
sns.relplot(x='model_year', y='mpg', col='origin', hue='cylinders',
    data=cars)
```

- `cars = sns.load\_dataset('mpg')`: loading the built-in MPG dataset that will be plotted.
- `type(cars)`: getting the type of the object cars output : `pandas.core.frame.DataFrame`
- `cars.shape`: plotting the data to watch the difference.
- `cars.dropna(inplace=True)`: removing all the rows in the dataset that contain missing values (NaN).
- `cars.head()`: getting the first five rows in the data set.
- `sns.relplot(x='model_year', y='mpg', col='origin', hue='cylinders', data=cars )`: for creating a plot (scatter plot) and for the full signature visit Seaborn replot documentation.

Figure 2: Fuel efficiency (mpg) over model year, separated by car origin plot



Seaborn: KDEplot

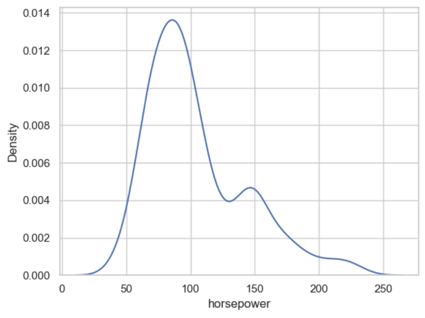
KDE (Kernel Density Estimation)

```
sns.kdeplot(data_to_be_plotted, fill=True, bw_adjust= value,
    cumulative=True)
```

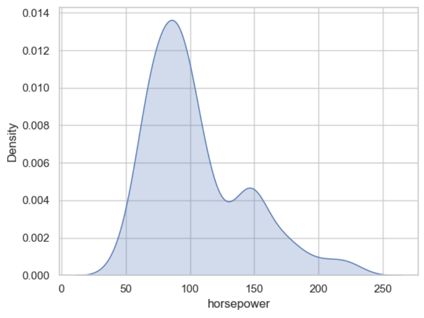
- `sns.kdeplot(data_to_be_plotted, fill=True, bw_adjust=value, cumulative=True`:
  - `sns.kdeplot()` for plotting KDE plot.
  - `fill` for filling the area under the curve.
  - `bw_adjust` for changing the bandwidth of the plot.
  - `cumulative = True` for making the plot a cumulative distribution function instead of a probability density function.

  `Note:` there are some changes in the parameters of this function. Visit Seaborn KDE plot documentation to see it and for more.

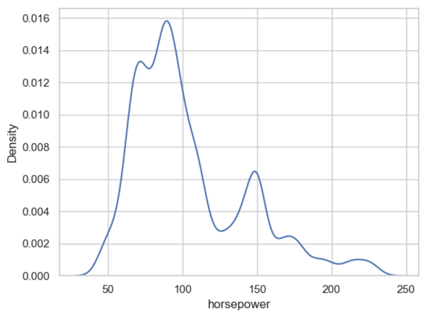
Figure 3: KDE Plots



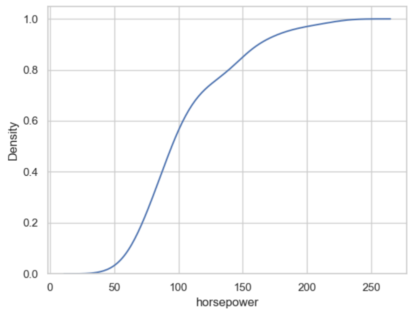
(a) default KDE plot



(b) Filled KDE plot



(c) KDE plot with a different bandwidth



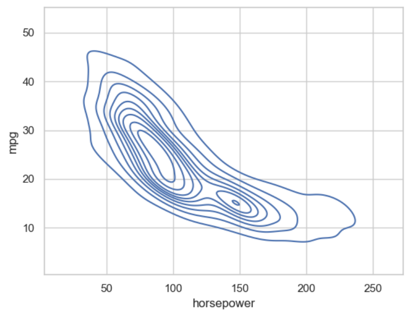
(d) KDE plot as a cumulative distribution function

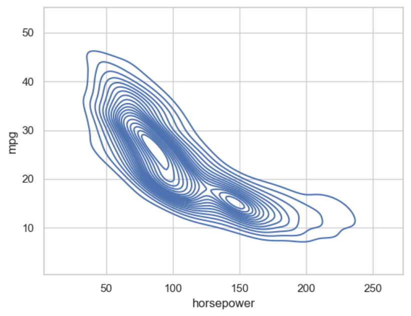
Bivariate KDEplot

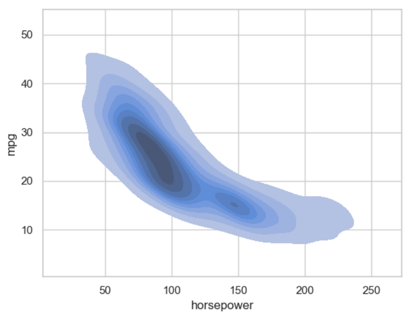
```
sns.kdeplot(x=data_1, y=data_2,levels=value, fill=True, cbar=True)
```

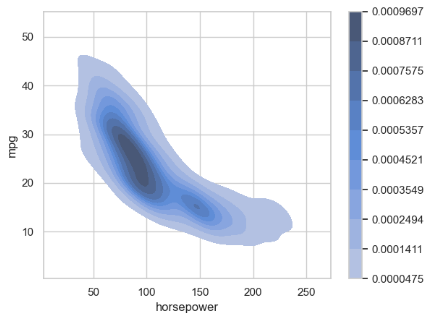
- `sns.kdeplot(x=data_1, y=data_2,levels=value, fill=True, cbar=True)`:
  - `sns.kdeplot(x=data_1, y=data_2)` for plotting bivariate KDE plot.
  - `levels` for changing the number of levels in the plot.
  - `fill` for filling the plot.
  - `cbar = True` for adding a color bar for the filled plot.

Figure 4: Bivariate KDE Plots


(a) Default bivariate KDE plot


(b) Bivariate KDE plot with changing levels


(c) Filled bivariate KDE plot

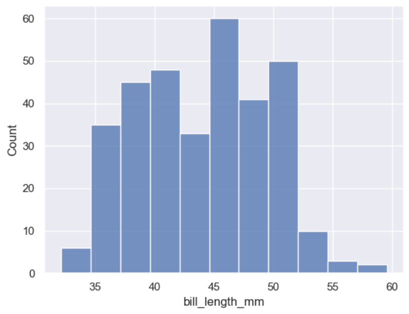

(d) Bivariate KDE plot with color bar

Hist plot

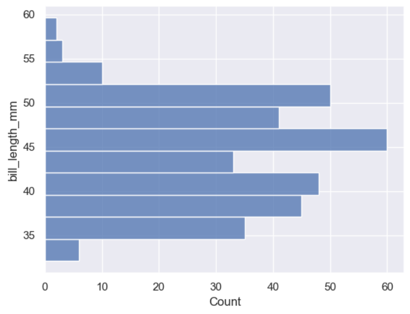
```
sns.histplot(penguins.bill_length_mm)
sns.histplot(x='bill_length_mm', data=penguins)
sns.histplot(y='bill_length_mm', data=penguins)
sns.histplot(x='bill_length_mm', data=penguins, kde=True)
sns.histplot(x='bill_length_mm', data=penguins, bins=20)
sns.histplot(x='bill_length_mm', data=penguins, binwidth=10)
sns.histplot(x='bill_length_mm', data=penguins, binrange=(30, 60))
sns.histplot(x='bill_length_mm', data=penguins, stat='density')
sns.histplot(x='bill_length_mm', data=penguins, stat='probability',
    cumulative=True)
sns.histplot(x='bill_length_mm', data=penguins, hue='species')
sns.histplot(x='bill_length_mm', data=penguins, hue='species',
    element = 'poly')
```

- `For more about Hist plot visit Seaborn Hist plot documentation`
- `sns.histplot(penguins.bill_length_mm)sns.histplot(x='bill_length_mm', data=penguins)`
- `sns.histplot(x='bill_length_mm', data=penguins)` : Both for plotting a histogram and visit Seaborn Hist plot documentation for more.
- `sns.histplot(y='bill_length_mm', data=penguins)` : for plotting a horizontal histogram.
- `sns.histplot(x='bill_length_mm', data=penguins,kde=True, bins=value, binwidth=value, binrange=(value_1, value_2))`:
  - `kde=True` for adding a KDE plot.
  - `bins = value` dividing the histogram into a desired number of bins.
  - `binwidth = value` choosing a specific bandwidth for the bin in the plot.
  - `binrange = (value1, value2)` dividing the histogram bins with a specific range.
  - `stat ='density', stat = 'probability'` for making the values on y a density representation or a probability representation, and see the documentation at the top for more.
  - `cumulative = True` represent a cumulative function.
  - `hue = ' '` splits the data sent by the hue column and colors each different species by a color.
  - `element = ' '` Visual representation of the histogram statistic.

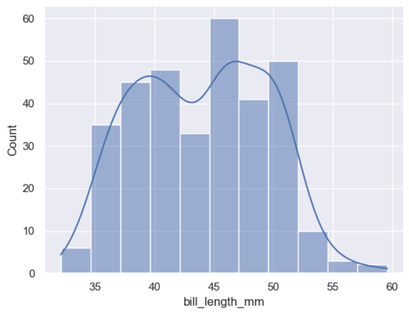
Figure 5: Histogram Plots – Part 1



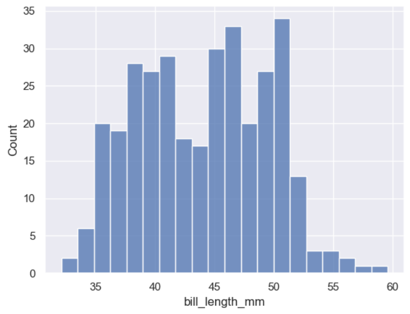
(a) Default Histogram plot



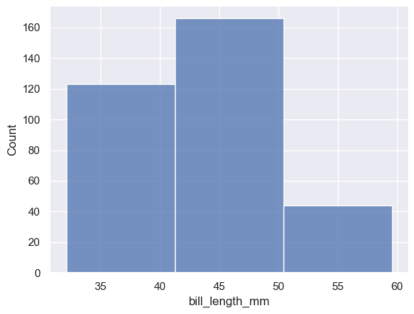
(b) Horizontal Histogram plot



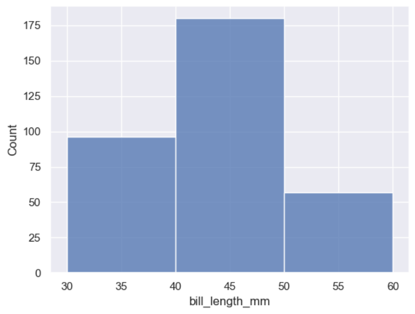
(c) Histogram plot with KDE plot



(d) Histogram plot with 20 bins

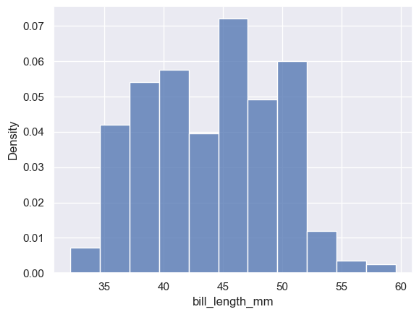


(e) Histogram plot with a specific bandwidth (10)

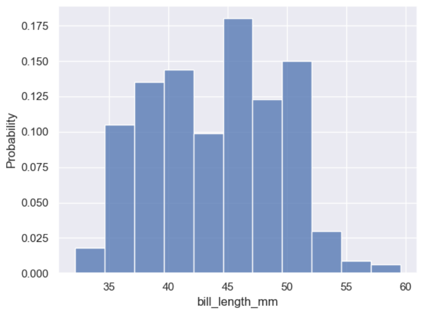


(f) Histogram plot with a specific binrange (30, 60)

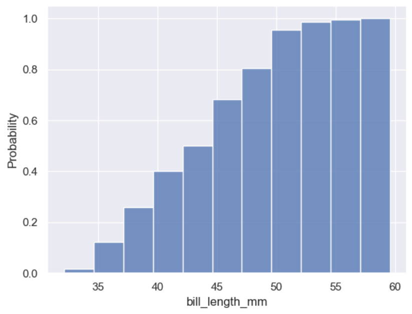
Figure 6: Histogram Plots – Part 2



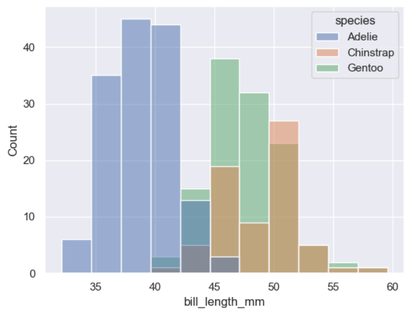
(a) Histogram plot with a density representation



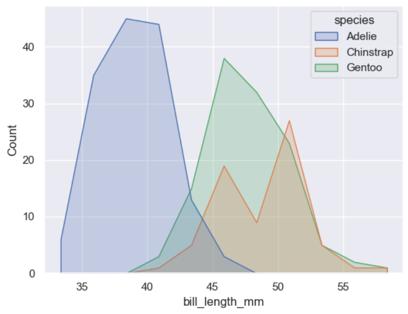
(b) Histogram plot with a probability representation



(c) Histogram plot with a probability cumulative representation



(d) Histogram plot with the hue



(e) Histogram visual representation using poly

ECDF plot

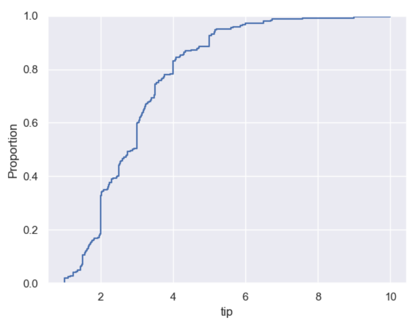
ECDF (Empirical Cumulative Distribution Function)

- For more about ECDF plot visit Seaborn ECDF plot documentation
- Advantage
  - No binning or smoothing
  - Compare Category distribution is easy

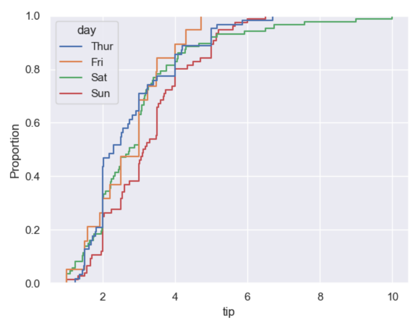
- Dis Advantage
  - Hard to see Central Tendecies
  - Hard to detect as we have a Bimodal distribution

```
# For plotting an ECDF plot
sns.ecdfplot(x='tip', data=tips);

#Using hue to split the plot
sns.ecdfplot(x='tip', data=tips, hue = 'day');

#Using state for changing the y representation.
sns.ecdfplot(x='tip', data=tips,stat = 'count', hue = 'day');

#Using lw for styling (changing line width)
sns.ecdfplot(x='tip', data =tips,hue='day', lw = 3);
```

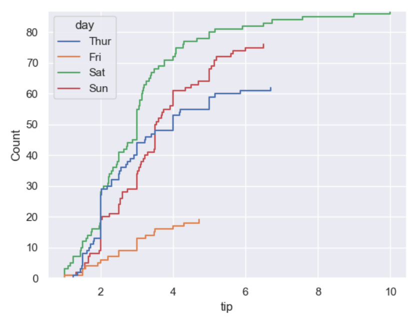
Figure 7: ECDF plots



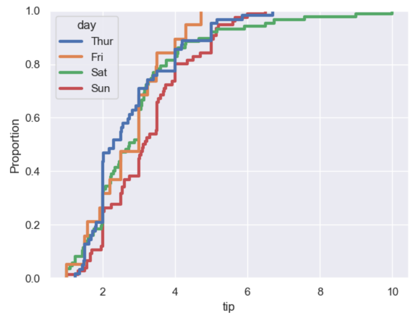
(a) Default ECDF plot



(b) ECDF plot using hue



(c) ECDF plot using stat and hue



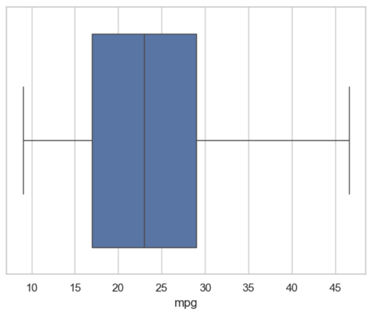
(d) ECDF plot with a styled line width

Box plot

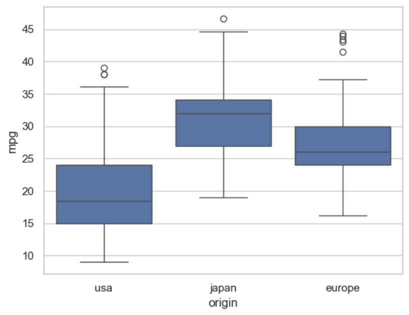
- Seaborn classifies the box plot as a categorical distribution plot
- For more about Box plot visit Seaborn Box plot documentation
- Note: you can find some styling that isn't mentioned in the Seaborn documentation, but it is in the Matplotlib documentation, so visit Matplotlib documentation

```
# For plotting a  Box plot
# (for a horizontal Box plot reverse the x and y and the reverse is
   true)
sns.boxplot(x= cars.mpg)
sns.boxplot(x= cars.origin,y = cars.mpg)
sns.boxplot(x= 'origin' ,y='mpg', data = cars)
sns.boxplot(y='origin', x='mpg', data=cars)


# For plotting a vertical Box plot
sns.boxplot(cars.mpg)
sns.boxplot(y=cars.mpg)


# Using hue to split the plot
sns.boxplot(x= 'origin' ,y='mpg', hue='cylinder', data = cars)


# Using some styling
sns.boxplot(x='mpg', y='origin', data=cars,hue='newer_model',
   linewidth=2.5, fliersize=5)
```

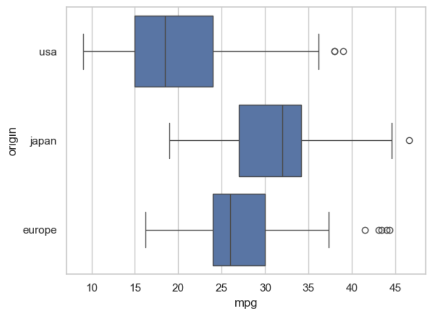
Figure 8: Box plots



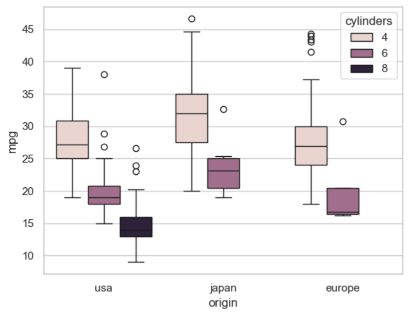
(a) Default Box plot



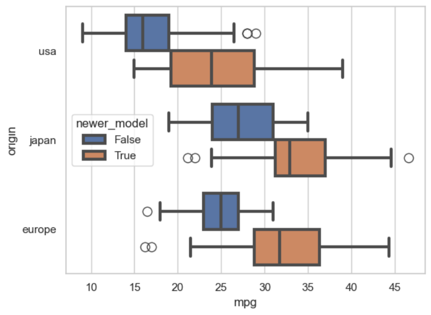
(b) Vertical Box plot with x and y data



(c) Horizontal Box plot with x and y data



(d) Box plot using hue to split data



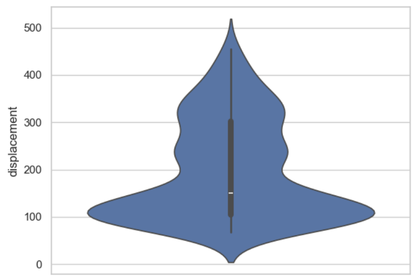
(e) Box plot with a bigger fliersize and linewidth

Violin plot

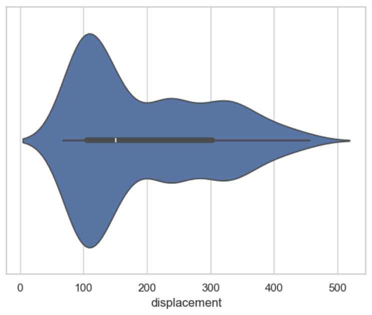
- Seaborn classifies the box plot as a categorical distribution plot
- For more about Box plot visit Seaborn Box plot documentation
- Note: you can find some styling that isn't mentioned in the Seaborn documentation, but it is in the Matplotlib documentation, so visit Matplotlib documentation

```
# For plotting a  Violin plot
sns.violinplot(cars.displacement)
sns.violinplot(x=cars.cylinders, y=cars.displacement);
sns.violinplot(x='cylinders', y='displacement',data=cars)


# For plotting a Horizontal Violin plot
sns.violinplot(x = cars.displacement)


# Using hue to split the plot
sns.violinplot(x='cylinders', y='displacement', hue='origin', data=
    cars);


# Using split to split the violin plot into parts for each category
    in the x variable.
sns.violinplot(x='cylinders', y='displacement', hue='origin', data=
    cars[cars.origin.isin(['japan', 'europe'])], split=True)

# Using some styling
sns.boxplot(x='mpg', y='origin', data=cars,hue='newer_model',
    linewidth=2.5, fliersize=5)


#changing the inner part of the violin plot
sns.violinplot(x='cylinders', y='displacement', hue='origin', data=
    cars[cars.origin.isin(['japan', 'europe'])], split=True, inner='
    quartiles')


# Using some styling like changing the bandwidth
sns.violinplot(x=cars.cylinders, y=cars.displacement, bw_method
    =0.2);
```

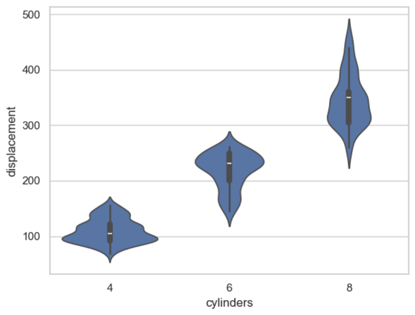
Figure 9: Violin plots



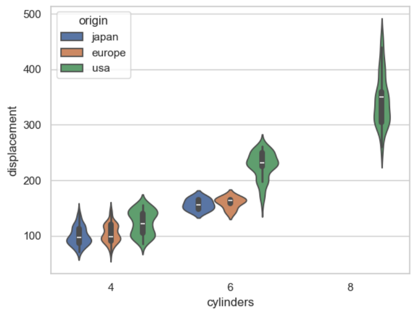
(a) Default Violin plot



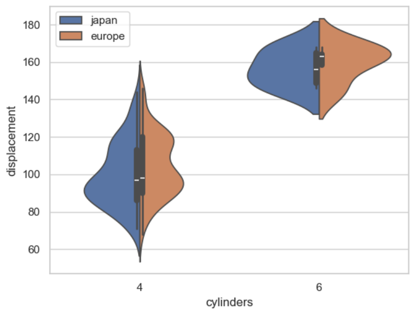
(b) Horizontal Violin plot



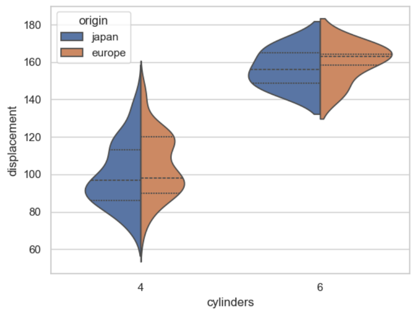
(c) Vertical Violin plot with x and y data



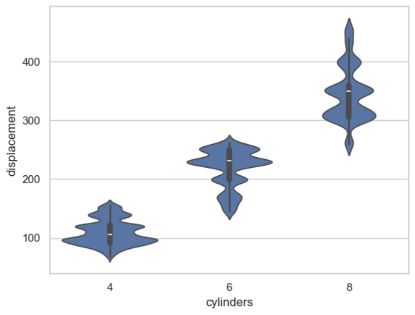
(d) Violin plot using hue to split data



(e) Violin plot using split



(f) Violin plot with changed inner part



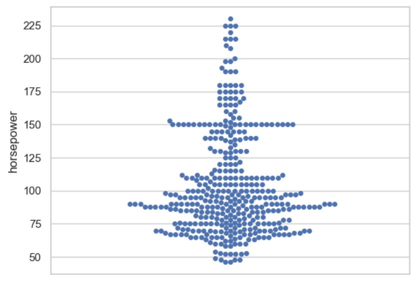
(g) Violin plot with changed bandwidth

Swarm plot

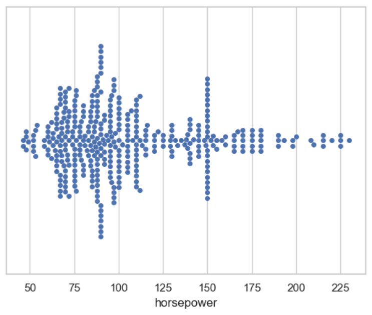
- In Swarm plot, there is no overlap
- For more about Swarm plot visit Seaborn Swarm plot documentation

```
# For plotting a  Swarm plot
# (For a horizontal Box plot reverse the x and y and the reverse is
    true)
swarmplot(cars.horsepower);
sns.swarmplot(x=cars.origin, y=cars.horsepower)
sns.swarmplot(x='origin', y='horsepower', data=cars)

# For plotting a Horizontal Swarm plot
sns.swarmplot(x=cars.horsepower)


# Using hue to split the Swarm plot
sns.swarmplot(x='origin', y='horsepower', hue='cylinders', data=cars
    )


# Using  dodge to  split the Swarm plot data horizontally
sns.swarmplot(x='origin', y='horsepower', hue='cylinders', data=cars
    , dodge=True)

# Changing Swarm plot marker
sns.swarmplot(cars.horsepower, marker= 'X')


# Plotting a swam plot and a Box plot
sns.boxplot(x=usa.cylinders, y=usa.horsepower)
sns.swarmplot(x=usa.cylinders, y=usa.horsepower)


# Plotting a Swarm plot and a Violin plot
sns.violinplot(x=usa.cylinders, y=usa.horsepower)
sns.swarmplot(x=usa.cylinders, y=usa.horsepower)
```

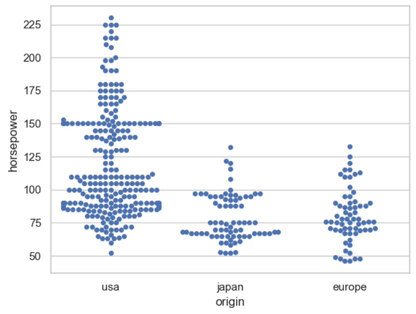
Figure 10: Swarm plots – Part 1



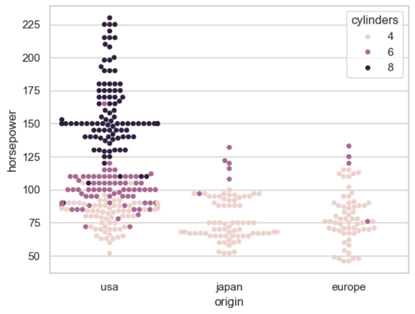
(a) Default Swarm plot



(b) Horizontal Swarm plot

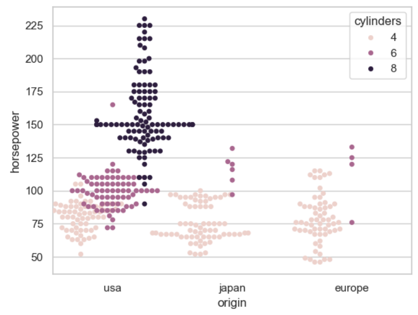


(c) Vertical Swarm plot with x and y data

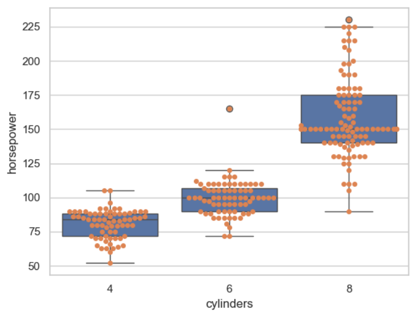


(d) Swarm plot using hue to split data

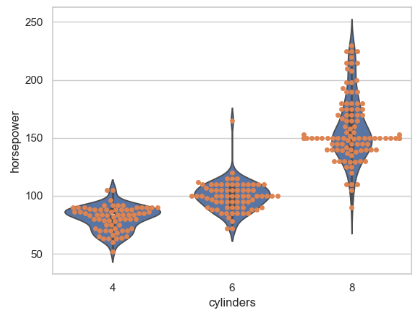
Figure 11: Swarm plots – Part 2



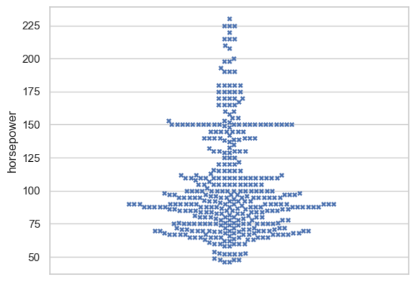
(a) Swarm plot using dodge to split data horizontally



(b) Swarm plot with Box plot



(c) Swarm plot with Violin plot



(d) Swarm plot with an X marker

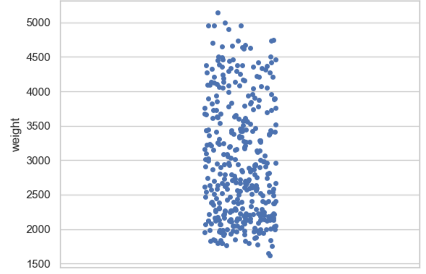
Strip plot

- For more about Strip plot visit Seaborn Strip plot documentation

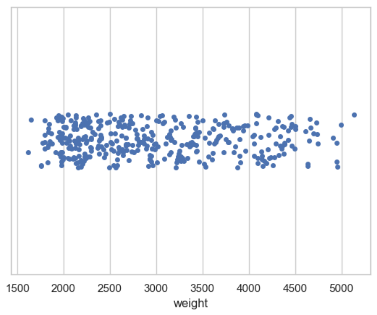
```
# For plotting a  Strip plot
sns.stripplot(cars.weight)
sns.stripplot(x=cars.weight, y=cars.origin)
sns.stripplot(x='weight',y='origin', data=cars)


# For plotting a Horizontal Strip plot
sns.stripplot(x=cars.weight)


# Using hue to split the Strip plot
sns.stripplot(x ='weight', y='origin', hue='cylinders', data=cars)


# Using  dodge to  split the Strip plot data horizontally
sns.stripplot(x ='weight', y='origin', hue='cylinders', data=cars,
    dodge=True)


# Changing spread using jitter
```

```
sns.stripplot(x=cars.weight, y=cars.origin,data=cars, hue='cylinders
    ', jitter=.25);


# Changing markers and size of markers
sns.stripplot(x=cars.weight, y=cars.origin,data=cars, hue='cylinders
    ', marker='*', size=7)


# Changing the transparency of the markers
sns.stripplot(x=cars.weight, y=cars.origin,data=cars, hue='cylinders
    ', size =7, alpha=0.25)
```

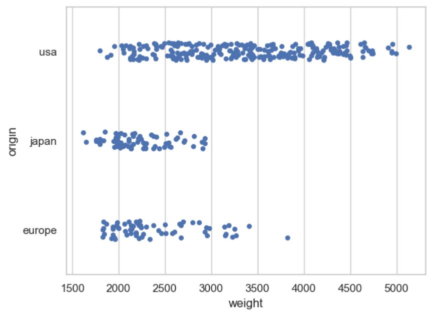
Figure 12: Strip plots – Part 1



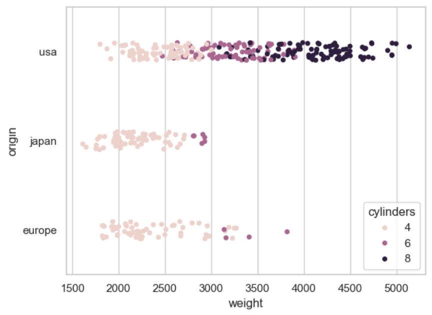
(a) Default Strip plot



(b) Horizontal Strip plot

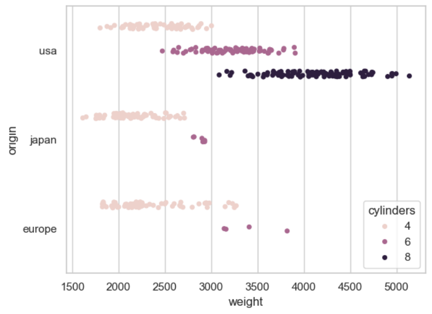


(c) Strip plot with x and y data

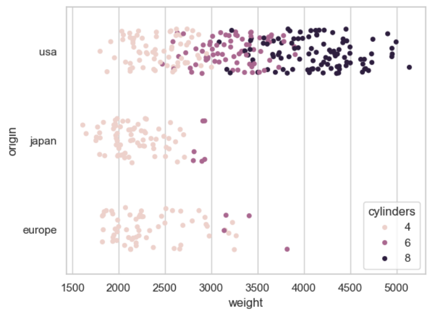


(d) Strip plot using hue to split data

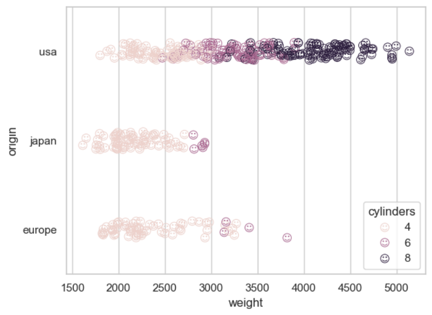
Figure 13: Strip plots – Part 2



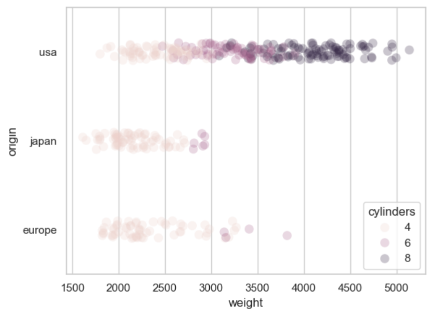
(a) Strip plot using dodge to split data horizontally



(b) Strip plot using jitter



(c) Strip plot with different markers and marker sizes



(d) Strip plot with alpha

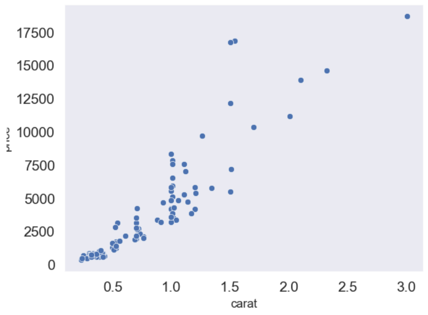
Scatter plot

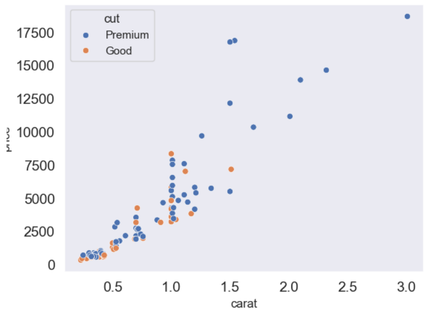
- Scatter plot is considered a rational plot
- For more about Scatter plot visit Seaborn Scatter plot documentation

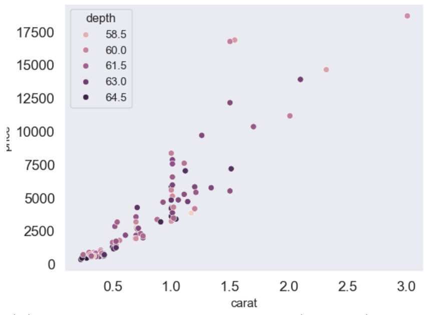
```
# For plotting a  Scatter plot
sns.scatterplot(x=diamonds.carat, y=diamonds.price)
sns.scatterplot(x='carat', y='price', data=diamonds)


# Using hue to split the Scatter plot
sns.scatterplot(x='carat', y='price', hue='cut', data=diamonds)
sns.scatterplot(x='carat', y='price', hue='depth', data=diamonds)

# Using  palette to  pass a specific color to the markers
sns.scatterplot(x='carat', y='price', hue='cut', data=diamonds,
    palette=['purple', '#55CCCC'])


# Using sizes to split the data  using marker size
sns.scatterplot(x='carat', y='price', size='cut', data=diamonds,
    sizes=[150, 50])
```

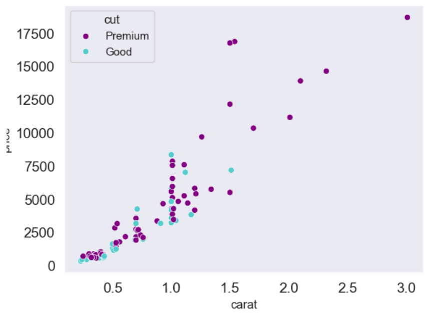
```
sns.scatterplot(x='carat', y='price', size='cut', data=diamonds,
    sizes=[150, 50])


# Using style and sizes to change markers and the size of markers
sns.scatterplot(x='carat', y='price', hue='cut', style='color', size
    ='depth', data=diamonds)
```

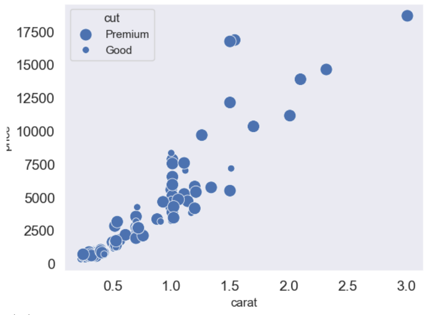
Figure 14: Scatter plots

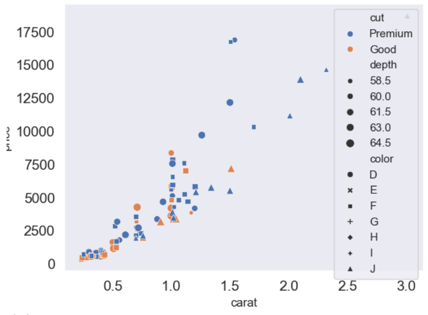

(a) Default Scatter plot


(b) Scatter plot using hue(cut) to Split data


(c) Scatter plot using hue(depth) to Split data


(d) Scatter plot with colors we made


(e) Scatter plot using sizes to Split data by marker size


(f) Scatter plot with splitted data by marker and size

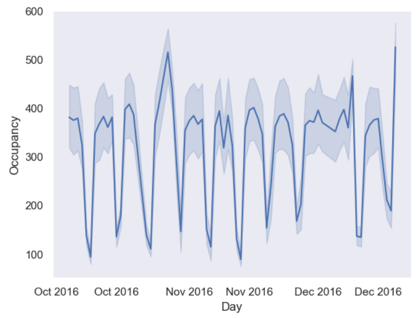
Line plot

- For more about Line plot visit Seaborn Line plot documentation

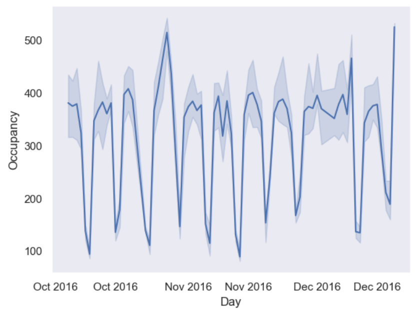
```
# For plotting a  Line plot
sns.lineplot(x=park.Day, y=park.Occupancy)
sns.lineplot(x='Day', y='Occupancy', data=park)


# Using n_boot to control the number of bootstrap  resamples to
    estimate uncertainty
sns.lineplot(x='Day', y='Occupancy', data=park, n_boot=1000)


# Using  ci to  change the confidence interval level for the line
    plot
sns.lineplot(x='Day', y='Occupancy', data=park, errorbar=('ci',70))


# Using an estimator changes the aggregate function for the data we
    want to visualize, without modifying the plot itself
sns.lineplot(x='Day', y='Occupancy', data=park, estimator='std')


# Using hue and style to split and change the style of the data
sns.lineplot(x='Day', y='Occupancy', data=park, hue="Location",
    style='Location')
```

- **Note:** To turn off the bootstrapped confidence intervals, set `ci=None` to trigger early exit within Seaborn code. A conditional checks for this case and completely bypasses the bootstrapping procedure if `ci` is set to `None`. This saves time if confidence intervals are not needed!

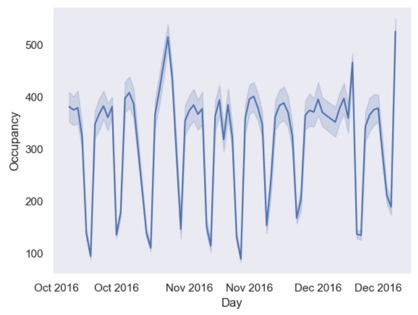
Figure 15: Line plots



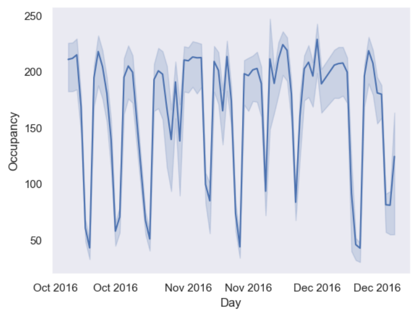
(a) Default Line plot



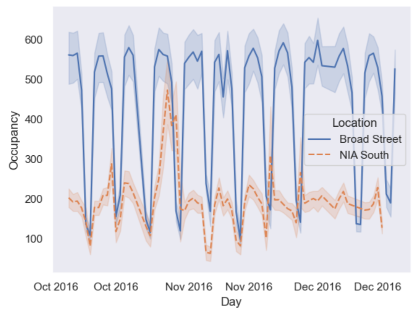
(b) Line plot with changing the number of bootstrap



(c) Line plot using ci to change confidence interval



(d) Line plot using estimator to make the aggregate function(std)



(e) Line plot using hue and style to Split and style data

Regression plot

- For more about Line plot visit Seaborn Regression plot documentation

```
# For plotting a  Reg plot
sns.regplot(x=diamonds.carat, y=diamonds.price)
sns.lineplotsns.regplot(x='carat', y='price', data=diamonds)
```

```
# Using fit_reg to remove the line and confidence interval
sns.regplot(x='carat', y='price', data=diamonds, fit_reg=False)


# Using  scatter to  remove scatter points
sns.regplot(x='carat', y='price', data=diamonds, scatter=False)


# Using  ci to  change the confidence interval level for the line
    plot  from (None, 100)
sns.regplot(x='carat', y='price', data=diamonds,ci = 70)


# Changing spread using  x_jitter
sns.regplot(x='cut_value', y='price', data=diamonds, x_jitter=.02)


#Using x_estimator for applying an aggregation function to the x-
    values before fitting the regression line
sns.regplot(x='cut_value', y='price', data=diamonds, x_estimator=np.
    mean)


#Using order to change the order of values fit a polynomial
    regression instead of a simple straight line (linear regression)
sns.regplot(x='carat', y='price', data=diamonds, order=2)

#Styling, for example, changing the line style, color, and width
sns.regplot(x='carat', y='price', data=diamonds, ci=None, line_kws={
    'lw': 4, 'color': 'black', 'linestyle': '-.'})
```

- **Note:** The `regplot` function allows customization using various parameters:
  - `order`: Controls the polynomial degree for fitting non-linear regression curves. For example, `order=2` fits a quadratic regression curve.

    ```
    sns.regplot(x='carat', y='price', data=diamonds, order
        =2)
    ```

  - `robust`: Applies robust regression, making the model more resistant to outliers, thus reducing the influence of extreme values on the fit.

    ```
    sns.regplot(x='carat', y='price', data=diamonds, robust
        =True)
    ```

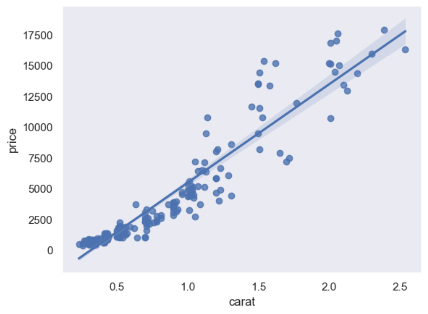
  - `logistic`: Fits a logistic regression (sigmoid curve), useful for binary outcome variables.

    ```
    sns.regplot(x='carat', y='price', data=diamonds,
        logistic=True)
    ```

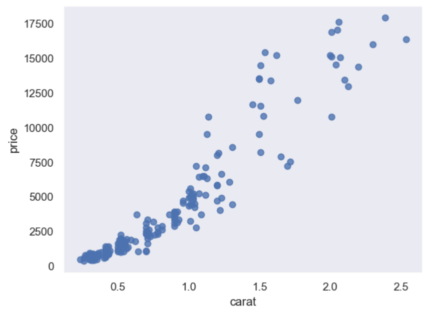
  - `lowess`: Applies locally weighted scatterplot smoothing (LOESS), fitting a smooth curve to the data without assuming a global functional form.

    ```
    sns.regplot(x='carat', y='price', data=diamonds, lowess
        =True)
    ```

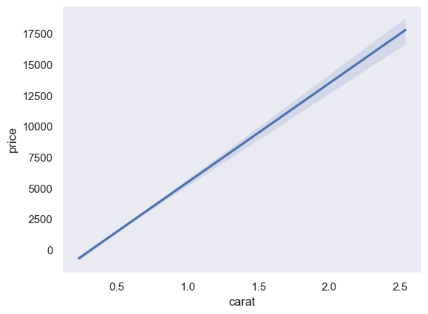
Figure 16: Regression plots – Part 1



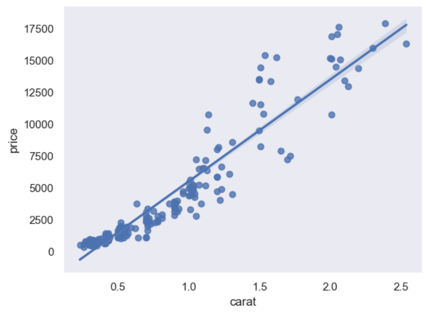
(a) Default Reg plot



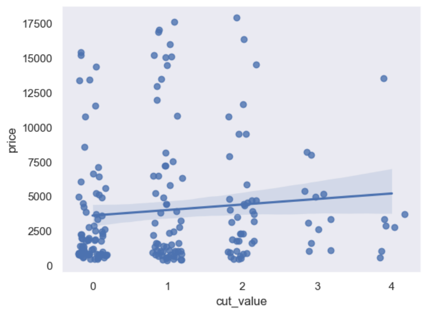
(b) Reg plot without the line and the ci



(c) Reg plot without the scatter points

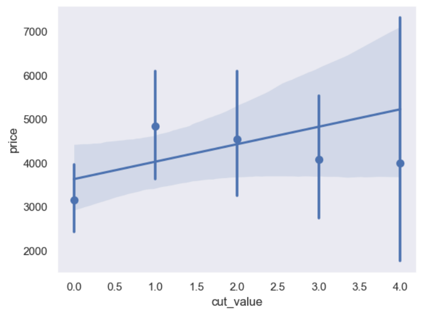


(d) Reg plot without the changed ci

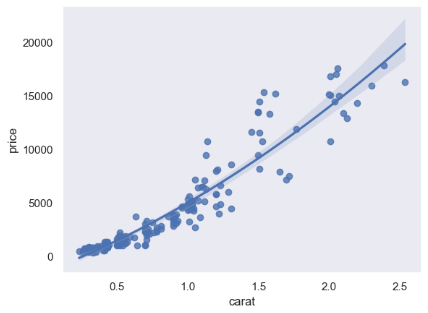


(e) Reg plot using x_jitter to chage spread

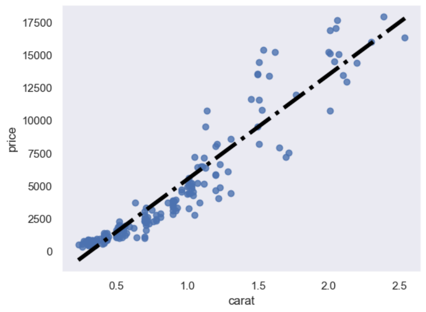
Figure 17: Regression plots – Part 2



(a) Reg plot using x_estimator for appling mean



(b) Reg plot with a polynomial regression



(c) Reg plot with changed line

Conclusion

At the end, this was a fast summary for the `Seaborn Library` by me,
and if it was useful, don't forget to pray for me `and for all our brothers in Palestine and Sudan`. And thank you :).