Module 2

Matplotlib

- Visualizing data is crucial to quickly understanding trends and relationship in your dataset.
- Matplotlib is one of the most popular libraries for plotting with python.

- Sometime known as "Grandfather of plotting and visualization libraries.
- Many visualization libraries are directly built ontop of matplotlib (e.g. seaborn and pandas built-in visualization).
- matplot is heavily inspired by the plotting function of matlab programming language.
- It allow creation of almost any plot type and heavy customization

- Two seperate approches to creating plots.
  1. functional plots.
  2. OOP based plots.

- Matplot Basic
  - Functional Methods
- Matplot figures and Subplots
  - OOP Method

Topic covered:-

- matplotlib basic and Function
- Malplotlib Figures
- Malplotlib subplots
- matplotlib styling
- Exercise Question and solution.

Two main goals with Matplotlib:

- Functional Relationship

$y = 2x$

- Be able to plot out a relationship between raw data points:

$x = [1, 2, 3, 4]$

$y = [2, 4, 6, 8]$

1) MatPlotLib Basic

- Basic way to use MatPlotLib is functional plot calls:
  - plt.plot(x,y)

- These function calls are really simple to use, but they don't actually allow high degree of use.
  - need plt.show() in .py scripts but in jupyter notebook Not needed.

Code

```
Import matplotlib.pyplot as plt
% matplotlib inline

Import numpy as np
x = np.arange(0, 10)
y = 2*x

plt.plot(x, y)
plt.show() — necessary in .py
```

↳ doesn't show the address on the memory.

```
[<matplotlib.lines.Line2D at 0x2066a>]

plt.plot(x, y)
plt.title("String Title")
plt.xlabel('x')
plt.ylabel('y')
```

add limits

```
plt.plot(x, y)
plt.xlim(0, 6)      ---> set limits on scale X
plt.ylim(0, 15)     ---> set limits on scale y
plt.savefig('myfirst.png')
```

Saving the plot:- (or provide the path)

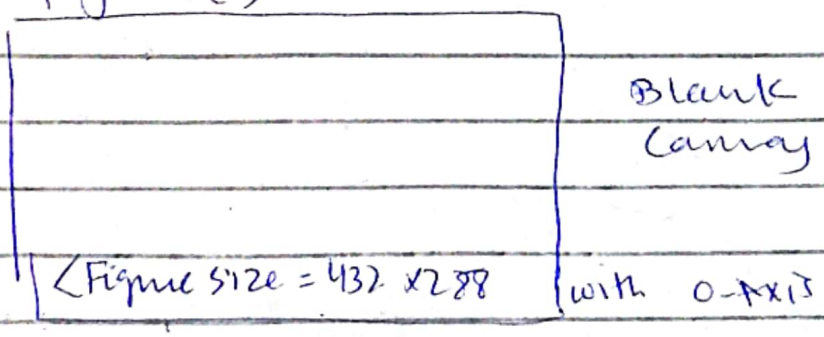
```
help(plt.savefig)
plt.savefig("C:\\Users\\Desktop\\hello.png")
```

03- Matplotlib - Understanding the Figure Object

part - One

Code # Invisible

```
plt.figure()
```




```
plt.figure(figsize=(10,10))
```

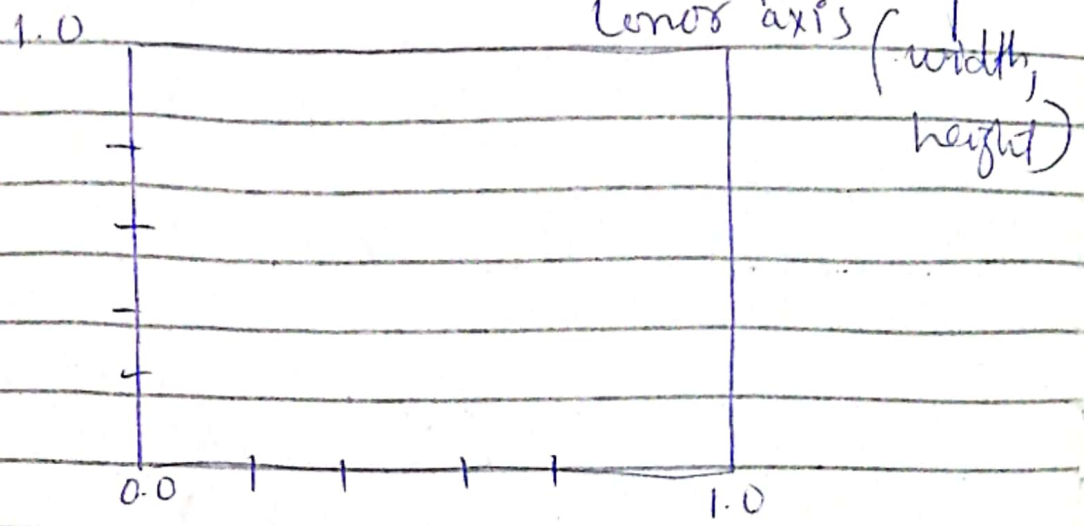
then

Figure size = 720 x 720

fig = plt.figure()

axes = fig.add_axes([0, 0, 1, 1])

[(x, y) low left corner axis (width, height)



to start

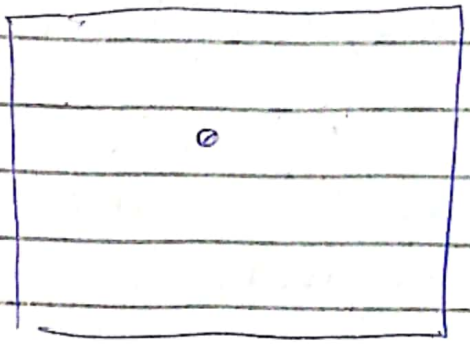


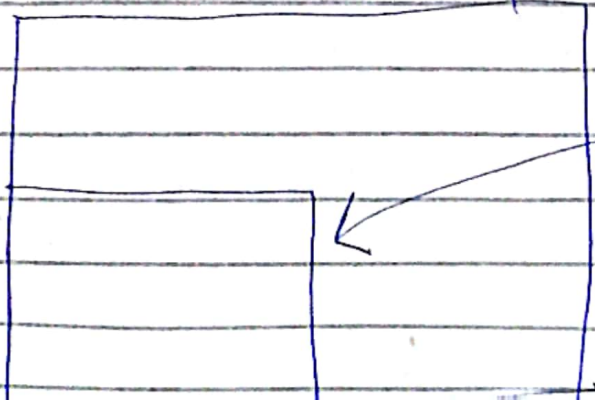
fig.add_axes([0.5, 0.5, 1, 1])

lower left corner of Axes

Figu = plt.figure()

axes = fig.add_axes([0, 0, 0.5, 0.5])

width and height



axes.plot(x, y) — plot the figure

4- Matplotlib - Implementing Figure and Axis part - two

```
a = np.linspace(0, 10, 11)
b = a**4
x = np.arrange(0, 10)
y = 2**x

fig = plt.figure()
axes = fig.add_axes([0,0,1,1])

axes.plot(x,y); | axes.plot(a,b)
plt.show()
```

Multiple axis on same figure #

```
fig = plt.figure( )
# Larges axis
axes1 = fig.add_axes([0,0,1,1])
axes1.plot(x,y)
# Small plot
axes = fig.add_axes([0.2,0.2,0.5,
                     0.5])
plt.show()
```

```
axes1.set_ylim(0, 50)
axes1.set_ylabel('B')
axes1.set_xlabel('Lable')
axes2.plot(x, y)
```

copy paste for axes2

we can plot zoom in in the figure

5. Matplotlib Figure Object part 3- Figure parameters.

```
Code #
fig = plt.figure()
axes1 = fig.add_axes([0, 0, 1, 1])
```

dpi = 200 → clear the image high resolution

```
fig = plt.figure(figsize=(10,3), dpi=200)
```

figsize → change figsize

dpi → clear resolution

```
fig.savefig('new-figure.png')
```

```
fig.savefig("image.png", bbox_inches='tight')
```

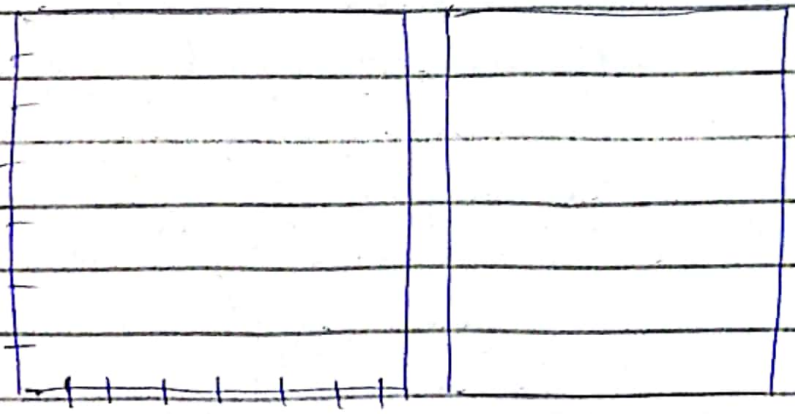
6- Matplot Subplots

- Subplot functionality

- The plt.subplot() call allow us to easily create a Figure and Axes objects. in side by side Formation -

- The plt.subplots() command returns a tuple containing the Figure canvas and then a numpy array holding the axes object.

fig, axes = plt.subplots(nrows=1, ncols=2)



- plt.subplots() return a tuple which by common convention we label (fig, axes):

- fig
  - This is is the entire figure canvas -
- axes
  - This is a numpy array

holding each of the axes according to position in the overall canvas.

Code #

```
import numpy as np
import matplotlib.pyplot as plt

a = np.linspace(0,10,11)
b = a**4

x = np.arange(0,10)
y = 2*x

fig, axes = plt.subplots(nrows=1, ncols=2)
axes[0].plot(x,y)
axes[1].plot(a,b)

fig, axes = plt.subplots(2,2)
axes[0][0].plot(x,y)
axes[1][0].plot(x,y)
```

loop through axes #

```
fig, axes = plt.subplots(3,1)
for ax in axes:
    ax.plot(x,y)
```

Everything too close together

at the end of

plt.tight_layout() — automatically attempt to space

```
fig, axes = plt.subplots(2, 2)
axes[0][0].plot(x, y)
axes[0][1].plot(x, y)
axes[1][0].plot(x, y)
axes[1][1].plot(x, y)
```

```
fig.subplots_adjust()
```

↓

wspace = 0.9, hspace = 0.9

```
axes[1][0].set_ylabel('Y label')
axes[1][1].set_title('Hello')
```

```
fig.suptitle("figure level", fontsize=16)
```

```
fig.savefig('new_subplot.png',
            bbox_inches='tight')
```

```
fig.set_figwidth(10)
```

7. Matplotlib Styling - lengends

part one: Legends

code #

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x,x)
ax.plot(x, x**2)

ax.plot(x, x**3, label = "X vs X^3")
ax.plot(x, x**2, label = "X vs X^2")
ax.legend()

ax.legend(loc="lower left")
ax.legend(loc=4)
```

4 → associate

```
ax.legend(loc=(1.1,0.5))
```

8- Matplotlib Syling :-

```
x = np.linspace(0,11,10)

fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x, x)
ax.plot(x, y, color='purple')
```

```
ax.plot(x, y, color = "#63ECS4")
ax.plot(x, x+1), color = 'purple', label =
                        "X vs X+1")
line width #
ax.plot(x, x+1, color = "#.e6c017",
                        Lw = 10)
Line width
ax.plot(x, x+1, color = "  ", linewidth = 0.5)
```

Linestyle #

```
ax.plot(x, y, linestyle = '-.')
ax.plot(x, y, linestyle = ":")
```

Custom linestyle #

```
lines = ax.plot(x, x+1, color = 'purple',
                        linewidth = 5)
lines[0].set_dashes([5, 2, 10, 2])
```

Markers :-

```
ax.plot(x, x+1, color='purple', lw=2,
        ls='--', markers='o')
```

markersize = 20 / ms = 20

```
ax.plot(x, y, color='purple', lw=2, ls="--",
        markers='o', markersize=20,
        markerfacecolor='red', markeredgewidth
        =8,
        markerfacecolor='orange')
```

9. Advanced Matplotlib Command (optional)

- matplotlib is a huge library!

Advance topics seperate jupyter file provided by the instructor.

Matplotlib Exercise

$E = mc^2$

1) m = eleven evenly space value

m = [0, 1, 2, 3, 4, 5, ...]

2) plot $E = mc^2$

x = Mass in grams
y = Energy of joules

3) plot the cell on logarithms

TWO #

Labels = ['1 Mo', '3 Mo', '6 Mo', '1 Yr',
'2 Yr', '3 Yr', '5 Yr', '7 Yr',
'10 Yr', '20 Yr']

July 16_2007 = [4.75, 4.98, 5.08 ....
JULY 16 - 2007 = [0.12, 0.11, 0.13,
0.14, 0.16, 0.17,
0.28, 0.46, 0.68, 1.37

Seaborn

- Seaborn is a statistical plotting library that is specifically designed to interexct well with pandas Dataframe to create common statistical plot types.
- Seaborn is built directly off Matplotlib, but uses a simpler "One-line" Syntax.
- Use Oneline function

```
sns.scatterplot(x='salary', y='sales',
                data = df)
```

Section Topics:

- Scatter plot
- Distribution plot
- Categorical plot
- Comparison plots
- seaborn plots
- Matrix plots

2- Scatterplot with Seaborn

- Scatter plot shows the relationship between two continous features.
- Recall that continous features are numeric variable that can take any number of values between any two values

- Continous features Examples:-
  - Age
  - Height
  - Salary
  - Temperature
  - prices

- Scatterplot line up a set of two continous features and plot them out as coordinates.

Code #

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (12,4))
sns.scatterplot(x='salary', y='sales', data=df)
```

either by continous or categorical

```
hue='Level of Education',
palette='Dark2',
size='salary'
s=200, alp
```

parameters in Scatterplot

x = 'sl' display on x axis

y = 'sw' display on y axis

data = df data which is used

hue = 'worker' differenciate color based on hue

size = ' ' size of scatter points changed based on size

s = 100 size of all scatter points

alpha = 1 — visibility of scatter points. — Transperency

style = 'level' — will set markers of different shape based on category

(can be used with hue = 'level' for better visualization)

palette = 'Dark2', check colormaps in matlab docs

plt.savefig("myplot.png")

plt.show() — in .py files

2. Distribution plots

part one

Understanding plot types

Distribution plots display a single continous feature and help visualize properties such as deviation and average value.

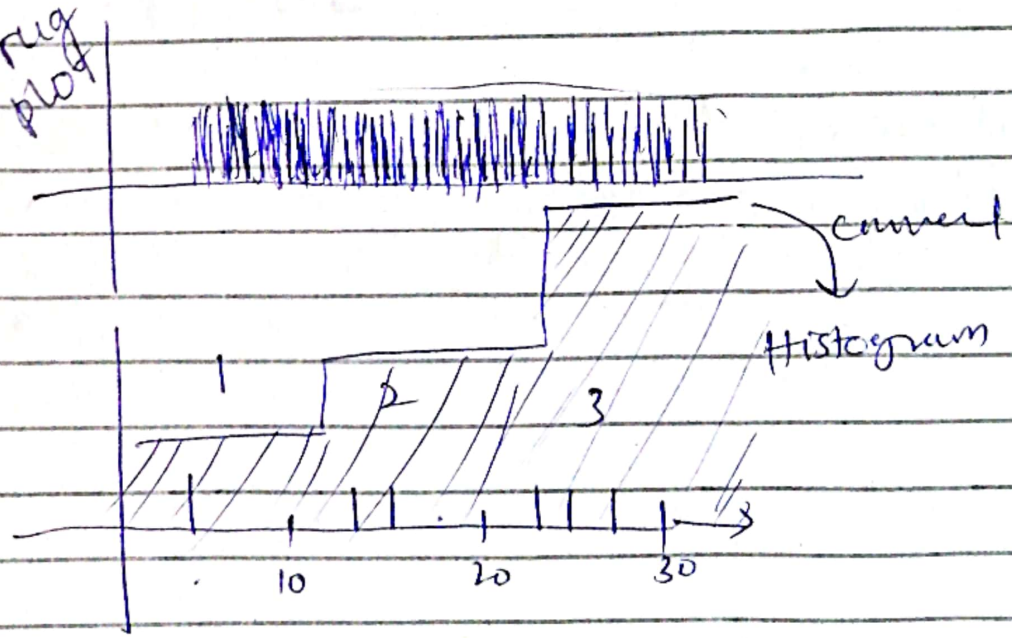
There are 3 main distribution plot types:

- Rug plot
- Histogram
- KDE plot

Rug plot :-

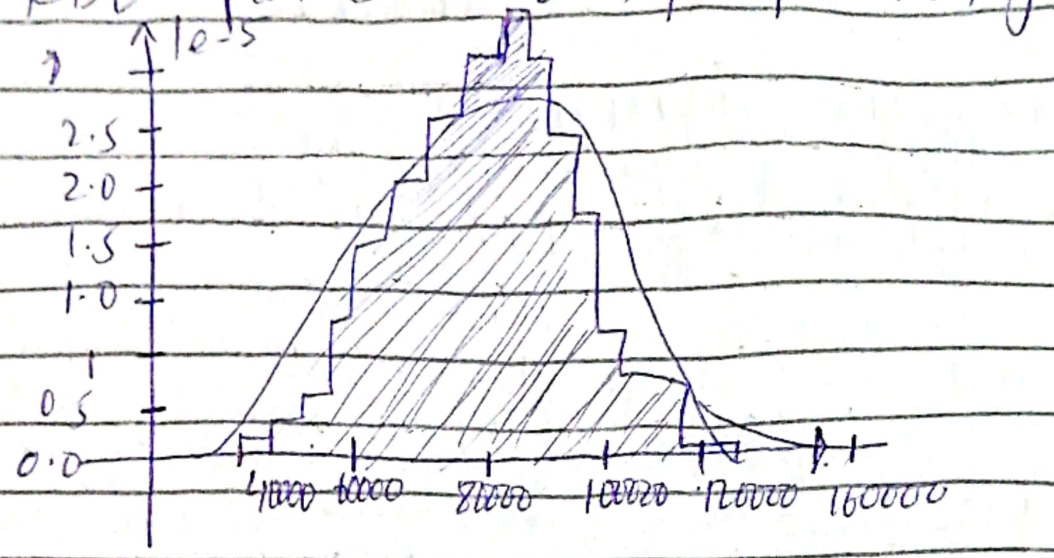
A rug plot is the simplest distribution plot and merely adds a dash or tick line for every single value.

The y-axis doesn't have any meanng



Y-axis can be seen as percent

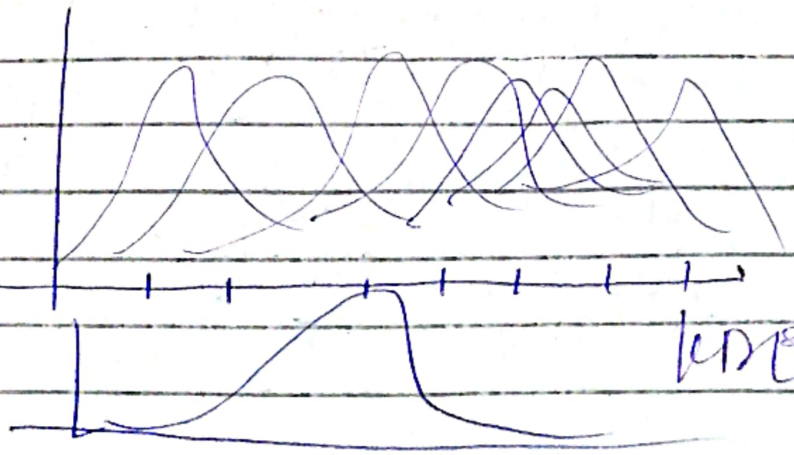
Seaborn allows us to add on a KDE plot curve on top of a histogram.



~~which show you what a continuous~~

- KDE stand for Kernel Density Estimation.
- Its a method of estimating a probability density function of a random variable. ~~variable~~ in simple terms, its way of estimating a continous probability curve for finite data sample.

Kernal = " Guasian distribution "



4- Distribution plot part-2

Code #

```
sns.rugplot(x='salarycolumn', data=df,
            height=0.5)

#Do not use DISTPLOT
sns.set(style='whitegrid')
sns.displot(data=df, x='salary', bins=10
```

Multiple color arm
color = "blue"
edgecolor = 'blue',
linewidth = '5',

kde = True
rug = True

```
sns.histplot(data=df, x='salary')
```

only KDE plot :-

```
sns.kdeplot(data=df, x='salary')
```

```
np.random.seed(42)
sample_ages = np.random.randint(0,100,20)
samples_ages = pd.DataFrame
```

```
sns.rugplot(data=df, x='age')
```

```
sns.displot(data=df, rug=True)
```

```
sns.kdeplot(data = sample, x = 'age',
            clip = [0, 100], bw_adjust
                              = 0.01
            shade = True)
```

clip = [0, 100]
bw_adjust = 0.01
shade = True

5. Categorical plot

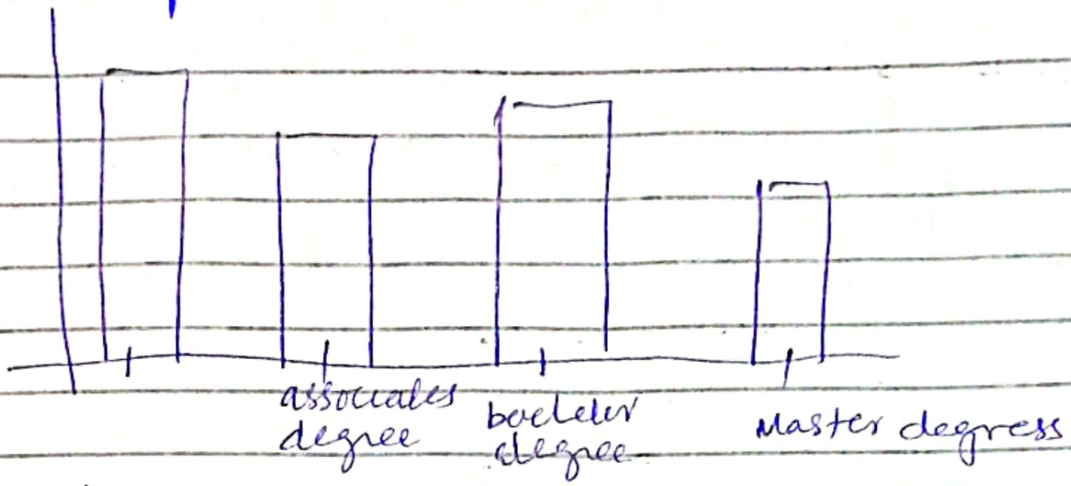
Statistical Estimation within categories -

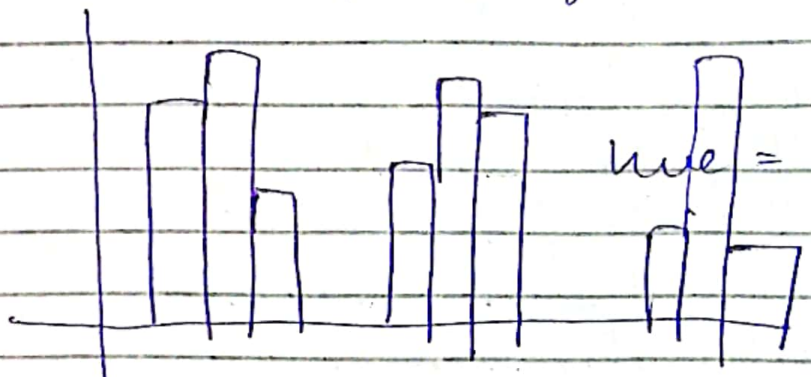
- The categorical plots discussed here will display a statistical mean value per category -
- for example mean value per category or a count of the numbers of rows per category -
- Its a visualization equivalent of a groupbyCall() -

Two main types of plot :-

- countplot()
  - Counts number of rows per category -
- barplot()
  - General form of displaying any chosen metrics per category -

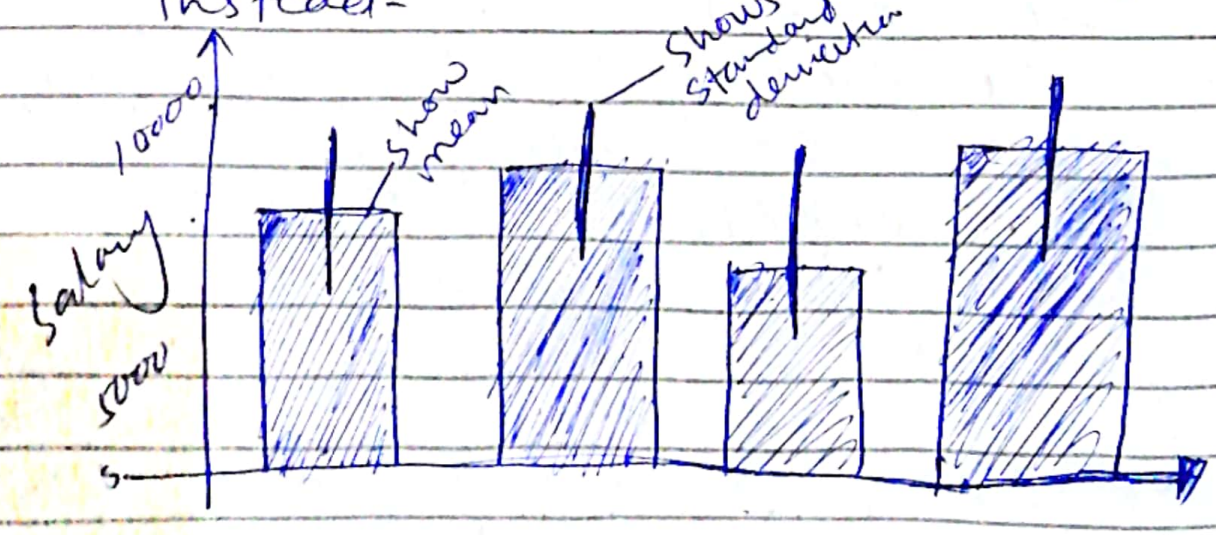
Count plot:-





Barplot :-

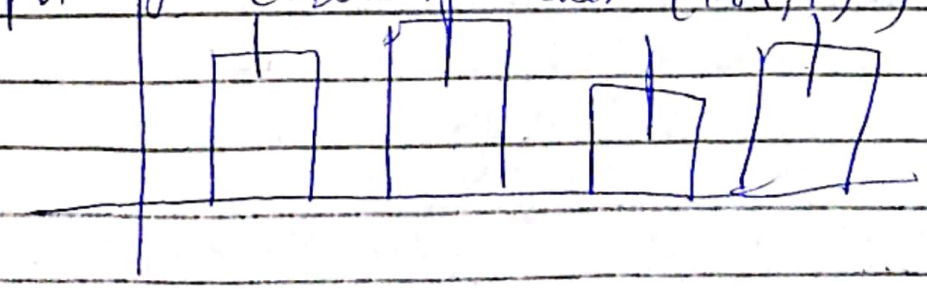
- The barplot is a general measure which allow us to chose any measure or estimator for the y-axis.
- we could plot the mean value and standard deviation per category instead-



6- Categorical plot :- part two

Statistical Estimation within Categories
part-two: Coding the plots

```
df.head()

df['division'].value_counts()
sns.countplot(data=df, x='division')
plt.xlim(100)  not good

sns.countplot(data=df, x='level of edu', hue='division')
                                  ↓ palette='Set2'

sns.barplot(data=df, x='level of edu',
            y='salary', estimator=np.mean,
                ci='sd')
            hue='division'
plt.legend(bbox_to_anchor=(1.05, 1))
```



007- Categorical plots-

Distribution within Categories

part = One

* Distribution within Categories
  - Boxplot
  - Violenplot
  - Swarmplot
  - Boxenplot (letter-Value point)

- Let explore understanding these plots on the previous salary datasets.

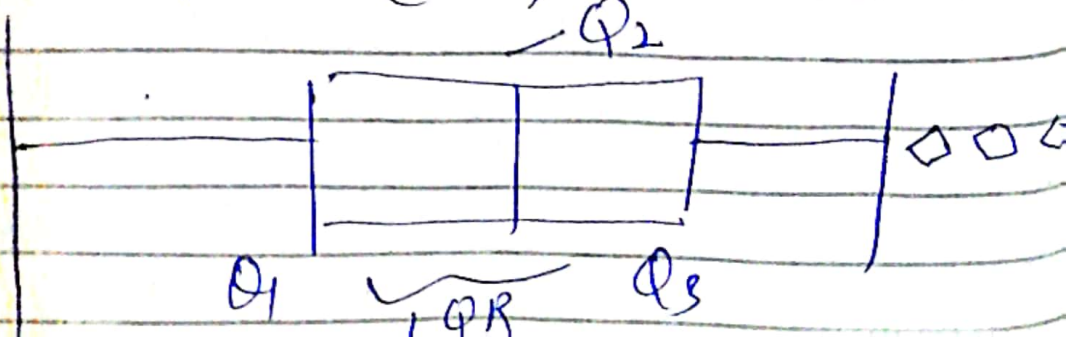
Boxplot :-

The boxplot displays distribution of a continous variable.

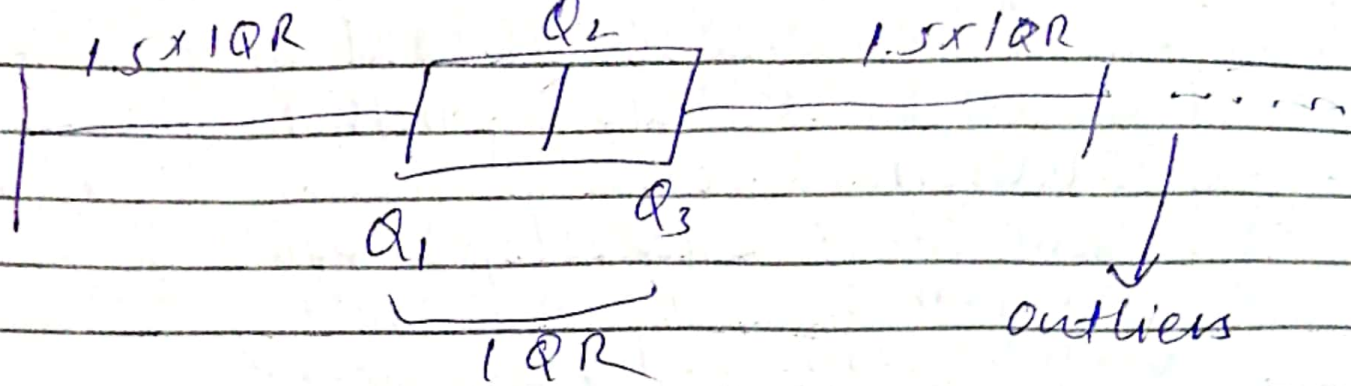
- It does this through the use of of quartiles.
- Quartiles seperate out the data into 4 ~~points~~ equal points.

number of data

- 25% of data points are in bottom quartile.
- 50th percent (Q2) is the median.

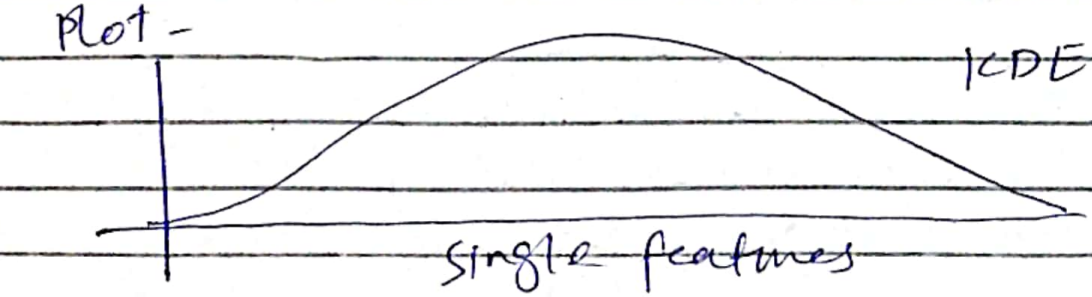


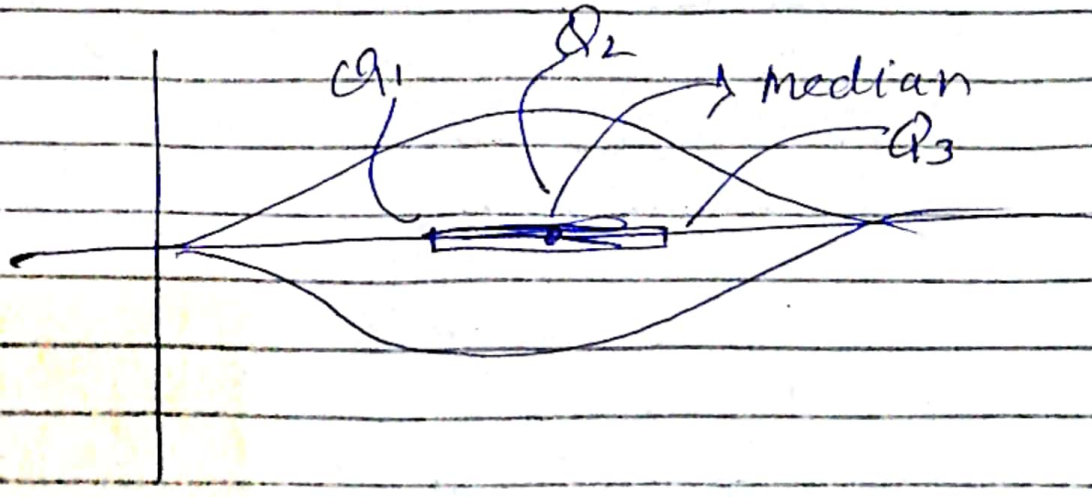
- Median is 50th percentile
- Median splits data in half-
- ~~IQR defines the box width~~



~~Violen~~ Violin plot :-

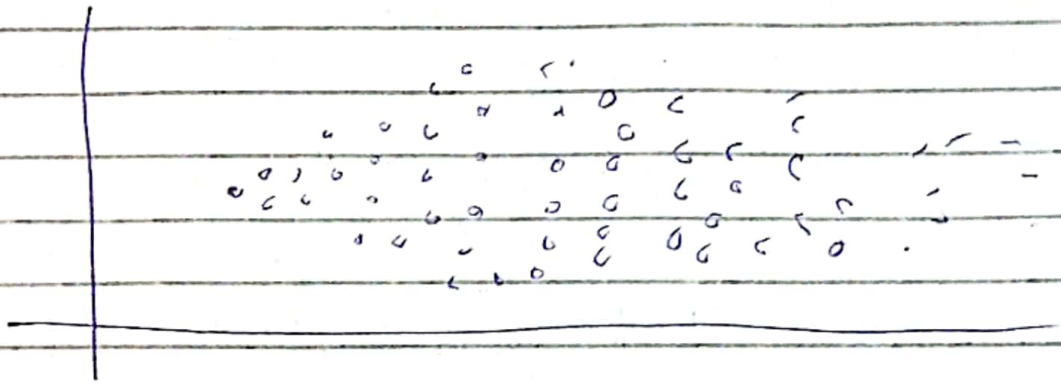
- The Violen plot plays a similar role as box plot -
- It display the probability density across the data using KDE.
- We can imagine it as a Mirror KDE Plot -





Swarmplot

- Swarmplot is very simple and simply shows all the data points in the distribution.
- for very large datasets, it won't show all the points, but will display the general distribution of them



will shows how many people we have per category

boxenplot (letter-value plot)

boxenplot is a relatively new plot developed in 2011 by ~~Dr~~ Heike Hofmann, Karen Kafadar and Hadley Wickham.

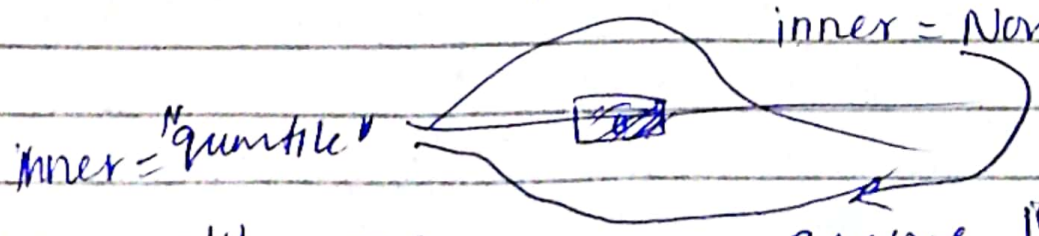
- Its mainly designed as an expansion upon the normal box plot.

- Make sure to read the linked paper in the notebook if you end up using this plot!

8 - Categorical plot - Distribution within categories.

Coding

```
df = pd.read_csv('StudentsPerformance.csv')
df.head()

sns.boxplot(data = df , y = 'math score')

sns.boxplot(data=df , y = 'math score',
            x = 'test preparation course')

            x = 'parental level'
            hue = 'test prepara'
plt.legend(bbox_to_anchor=(1,0.5,0.4))

violenplot()

sns.violinplot(data=df ,x = 'reading score',
               y = 'parental level of education',
               hue = 'test preparation course',
               palette='Set2')

               split = True,
               inner = None
```



```
inner = "quartile"

quartiles = True
```

Remove inner box plot -

bw = ——bandwidth stack in Gaussian kernel
(noise smaller value)
(low not good plot larger values)

```
plt.figure(figsize=(8,4), dpi=200)
sns.swarmplot(data=df, x='mathscore',
                                size=2
              y='gender',
              hue='test preptm
                   ourse')
```

dodge ~~dodge~~
dodge = & True
↳ split the hue to two graph

```
sns.boxenplot(x="math.score", y='test prepcourse',
              data=df, hue='gender')
```

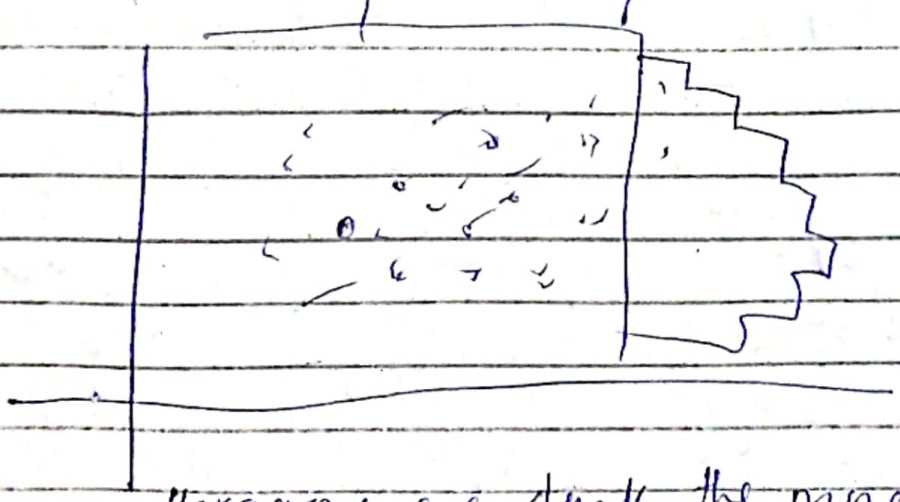
9. Comparison plots

part one = Understanding the plots-

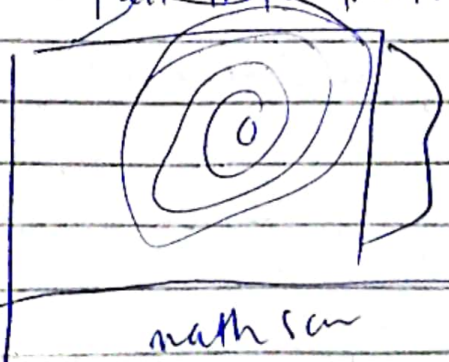
- Comparison plots are essentially 2-dimensional versions of the plots we've learned about so far
- The two mainplots types discussed here:
  - jointplot()
  - pairplot()

jointplot :-

- We can map histogram to each feature of a scatterplot to clarify the distribution within each feature.
- We can also adjust the scatterplot to be hex plot or a 2D KDE plot.

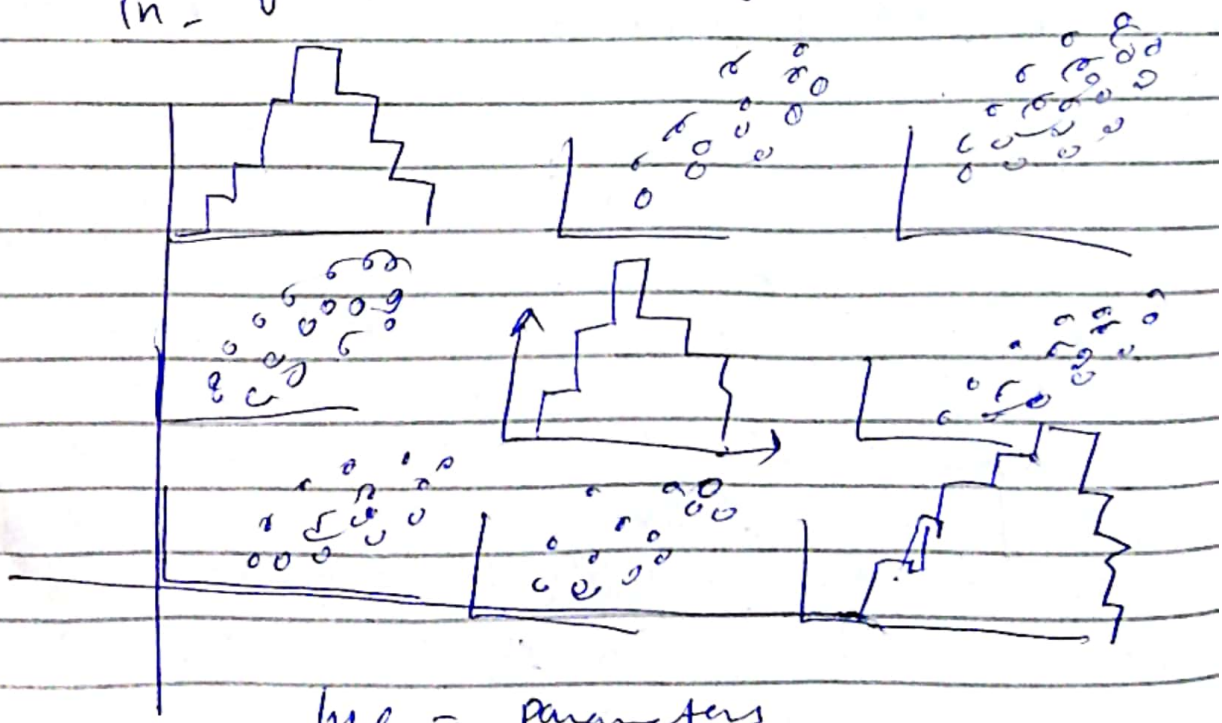


Hexagons are dark the more points fall into their area



- pairplot()
  - The pairplot is ~~superuseful~~. IS a quick way to compare all the numeric columns in a DataFrame.
  - It automatically creates a histogram for each column and a scatterplot comparison between all possible combination of columns.

Warning!
- pairplot() can be CPU and RAM intensive for large DataFrame with many columns.
- Its good idea to first filter down to only the columns you are interested in.



hue = parameters

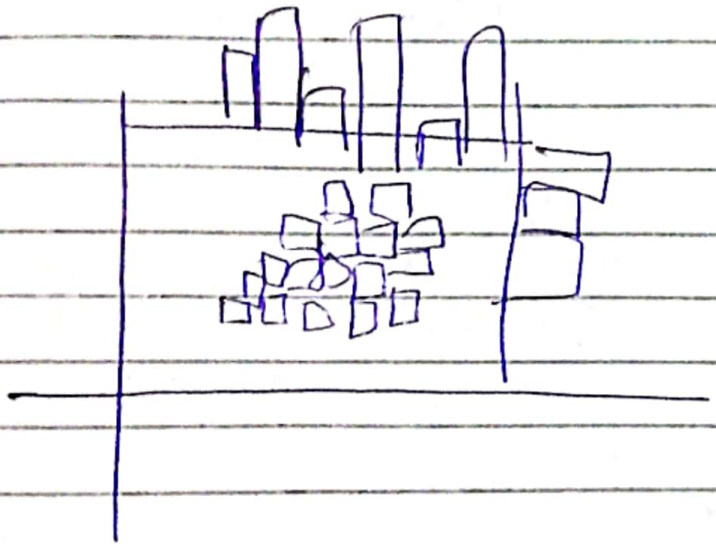
10- Comparison plots

part two = coding the plots

df.head()

sns.jointplot(data=df, x='math score',
y='reading score')

kind='scatter'
kind='hex'
alpha=0-1

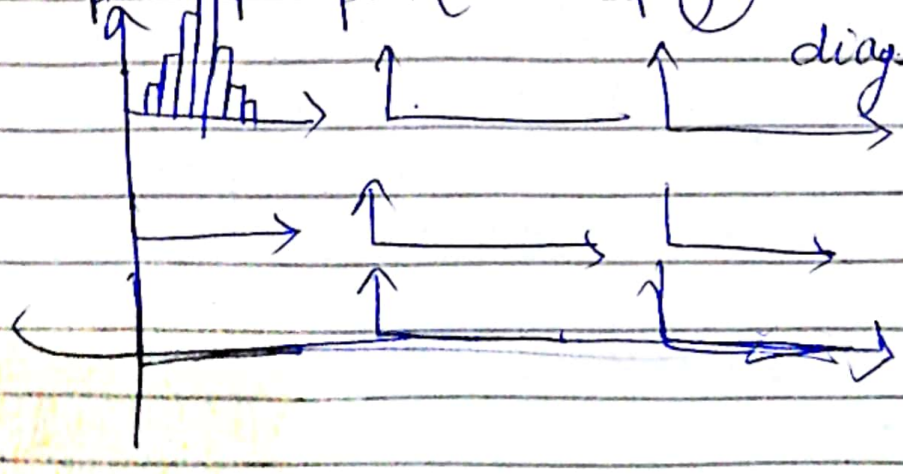


kind='hist'

kind='kde'
shade=True

hue='gender')

sns.pairplot(data=df, hue='gender',
diag_kind='hist')



sns.pairplot(data = df, hue = 'gender', corner = True)