Statistics:

Descriptive	Inferential
• Measure of Central Tendency ○ Mean ○ Median ○ Mode • Measure of Dispersion ○ Variance ○ Standard Deviation	Get inferences from the sample data using experiments like Z test, T test etc. and conclude about the Population where the sample have taken.

Population Mean: μ = $\sum_{i=1}^{N} \frac{x_i}{N}$

Population Variance: $\sigma^2 = \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{N}$

**Variance, how data disperse between data points*

Population SD: $\sigma = \sqrt{\sigma^2}$

**SD, how far the data point away from mean*

Sample Mean: $\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$

Sample Variance: $s^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n-1}$

** why n -1, to avoid underestimate the population variance so that $\bar{x} \approx \mu$ and $s^2 \approx \sigma^2$, this is called Bassel correction or degree of freedom n-1*

Sample SD: $s = \sqrt{s^2}$

Variable: Variable is a property that can take up any value

Quantitative Variable
- Discrete (No. of Students:4)
- Continuous (age: 4.7, height: 10.2)

Qualitative / Categorical
- Gender (Male, Female)
- Color (Blue, Green)

Random Variable: Values derived out of an experiment through some function

Discrete Random
- Tossing a Coin
- Rolling a Dice

Continuous Random
- How many inches it's going to rain
- Height of the people attending event

Percentile and Quartile:

Percentage
What is the percentage of odd numbers
{1,2,3,4,5,6) = 3/5*100 = 50%

Percentile
What is the percentile of value 9 of {1,2,2,3,4,5,5,6,7,8,8,8,9,9,10}
Percentile of value 9 = # of values below 9/n *100 = 78.57%
78.57% of entire distribution less then this value 9

To get the value when percentile given (find the 25% of the value)
Value = percentile/100 * (n+1) = 25/100(15) = 3.75

Quartile
- 25% = 1st Quartile
- 75% = 2nd Quartile
- 75% = 3rd Quartile

Number Summary:

{1,2,2,2,3,3,4,5,5,5,6,6,6,6,7,8,8,9,27}
- Min
- 1st Quartile (25%)
- Mean
- 3rd Quartile (75%)
- Max

IQR – Inter Quartile range
IQR = 3rd Quartile - 1st Quartile

Lower Fence = Q1 – 1.5(IQR)
Higher Fence = Q3 – 1.5(IQR)

* any value lower than lower fence or higher than higher fence is consider as outlier
* Boxplot will help to visualize outlier

Q1 = 25 percentile/100(n+1) = 25/100(20) = 5th position, which is 3
Q3 = 75/100(20) = 15th position which is 7
IQR = 7 – 3 = 4

Lower Fence = 3 – 1.5(4) = -3
Higher Fence = 7 - 1.5(4) = 13

Covariance and Correlation:
Covariance and Correlation are two statistical measures used to determine the relationship between two variables. Both ae used to understand how changes in one variable are associated with changes in another variable.

Covariance
Covariance is a measure of how much two random variables change together. If the variables tend to increase and decrease together, the covariance is positive. If one tend to increase when the other decreases, the covariance is negative.

Covariance is $cov\,(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x) * \left(y_i - \mu_y\right)$, Helps to find the direction of the relationship.

Pearson Correlation Coefficients $P_{(x,y)} = \frac{cov\,(x,y)}{\sigma_x\,\sigma_y}$, Helps to find the Strength and the Direction of the relationship, and the range $P_{(x,y)}$ between -1 to +1.

Exp: Size of Home Vs Price (positive and negative relationship)

Cov(x,y) = $\sum_{i=1}^{n}\frac{(x_i - \bar{x})(y_i - \bar{y})}{n-1}$ and Cov(x,x) = Variance(x,x) ie., $s^2 = \sum_{i=1}^{n}\frac{(x_i - \bar{x})^2}{n-1}$

$x_i - data\ point\ of\ random\ variable\ x$
$\bar{x} - sample\ mean\ of\ x$
$y_i - data\ point\ of\ random\ variable\ y$
$\bar{y} - sample\ mean\ of\ y$
$n - total\ number\ of\ sample$

Advantages
- Quantify the Relationship between X and Y

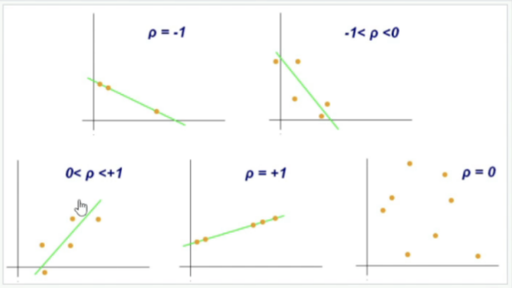
Disadvantage
- Covariance does not have a specific limit value ie., it won't' tell how strong the relationship Cov(x,y) => -$\alpha\ to + \ \alpha$

Correlation

Pearson Correlation Coefficient
* It limits the value between -1 to 1

$$\rho_{x,y} = \frac{cov(x,y)}{\sigma_x\,\sigma_y}$$

- The move the value towards +1 the more +ve correlated x and y
- The more the value towards -1 the more -ve correlated x and y

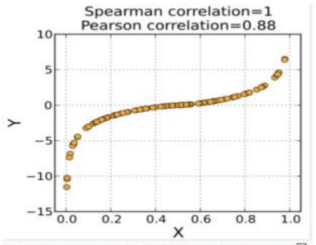


Examples of scatter diagrams with different values of correlation coefficient (ρ)

Spearman Rank Correlation
* Pearson Correlation Coefficient works better for linear data but not non-linear data
* To work with non-linear data, use Spearman Rank Correlation

$$r_s = \frac{cov(R_x, R_y)}{\sigma(R_x) * \sigma(R_y)}$$

$$R_x, R_y - Rank\ of\ x, y$$



A Spearman correlation of 1 results when the two variables being compared are monotonically related, even if their relationship is not linear. This means that all data points with greater x values than that of a given data point will have greater y values as well. In contrast, this does not give a perfect Pearson

Use Case
In ML Model Feature selection, when any of the feature correlation coefficient is ≈ 0 when compare with label value.

Probability: It's about determining the likelihood of an event or experiment

Toss a coin {H,T}
Pr(H) = ½ = 50%
Pr(T) = ½ = 50%

Rolling a dice {1,2,3,4,5,6}
Pr(x=1) = 1/6

Mutual Exclusive Event (Additive rule)
Two events are Mutually Exclusive if they cannot occur at the same time
Exp: Tossing a Coin, Rolling a dice

Non Mutual Exclusive Event (Additive rule)
Two Mutual Event can occur at the same time
Exp: Taking a card 'K' from a deck, K can have K and K Hearteen

Pr(H or T) = Pr(H) + Pr(T) *(Additive rule for mutual exclusive event)

Pr(1 or 5) = Pr(1/6) + Pr(1/6) = 1/3

Multiplication Rule (independent and dependent event)
Two events are ***independent*** if they do not affect one another
Exp: Tossing a coin and Rolling a dice

Pr(H and T) = Pr(H) * Pr(T) =1/2 * 1/2 = 1/4

Pr(K,H) = Pr(K) + Pr(H) – (Pr(K) and Pr(H))
= 4/52 + 13/52 – 1/52 = 16/52

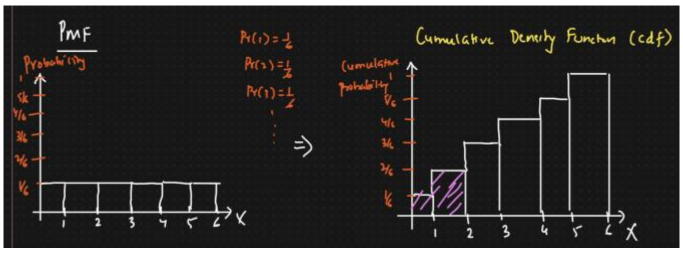
Two event are ***dependent*** if they affect one another
Exp:Take a King from a deck and then take Queen from a deck
Pr(K) = 4/52   Pr(Q) = 4/51

Pr(K and Q) = Pr(K) * Pr(Q/K) **** Conditional Probability***

Probability Distribution Functions:

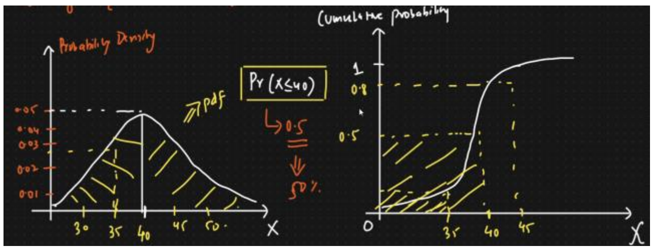
Probability Distribution Functions describe how the probabilities are distributed over the values of random variables.

Probability Mass Function (PMF)
Used for discrete random variables



Pr(x ≤ 2) = Pr(x = 1) + Pr(x = 2)
Pr(1/6) + Pr(1/6) = 1/3

Probability Density Function (PDF)
Used for continuous random variables



* PDF is a ***gradient*** (slope) of cumulative density function

* PDF Properties

- Non Negative f(x) ≥ 0 for all x
- The total area under the PDF cure is equal to 1 $\int_{-\infty}^{+\infty} f(x)\ dx = 1$

Types of Probability Distribution:

- Bernoulli Distribution – out come are binary (PMF), discrete random variable
- Binomial Distribution – PMF
- Normal / Gaussian Distribution – PDF, continuous random variable
- Poisson Distribution - PMF
- Log Normal Distribution – PDF
- Uniform Distribution – PMF

Bernoulli Distribution:

The Bernoulli Distribution is the simplest discrete probability distribution. It represents the probability distribution of a random variable that has exactly two possible outcomes, success (with probability p) and failure (with probability 1-p). It is used to model binary outcomes, such as coin flip or a yes/no questions.

- Discrete Random Variable (PMF)
- Outcomes are Binary

Exp: Tossing a coin
Pr(x=H) = 0.5 = p
Pr(x=T) = 1-0.5 = 0.5 = q

Whether the person will pass or fail
Pr(x=Pass) = 0.4
Pr(x=Fail) = 1-0.4= 0.6

PMF = $p^k(1-p)^{1-k}$
Pr(k=1) = $p^1(1-p)^{1-1}$ = p
Pr(k=0) = $p^0(1-p)^{1-1} = 1 - p = q$

simplified pmf $\begin{cases} q = 1 - p & if\ k = 0 \\ p & if\ k = 1 \end{cases}$

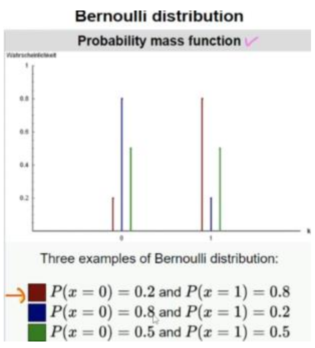
Mean of Bernoulli Distribution (w.r.to pic)
E(x) = $\sum_{i=0}^{1} k * p(k)$
= 0 * 0.4 + 1 * 0.6 = 0.6 => p

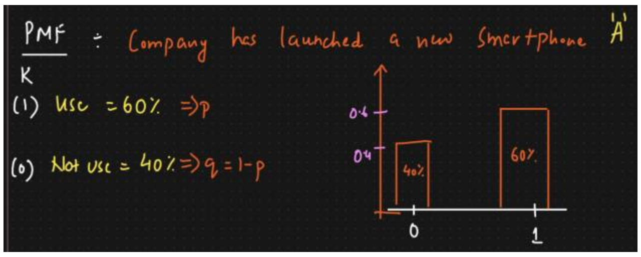
Median of Bernoulli Distribution (w.r.to pic)

Median $\begin{cases} 0 & if\ p < \frac{1}{2} \\ [0,1] & if\ p = \frac{1}{2} \\ 1 & if\ p > \frac{1}{2} \end{cases}$

Mode of Bernoulli Distribution (w.r.to pic)
p > q then p will be mode else q will be mode

Variance, SD of Bernoulli Distribution (w.r.to pic)





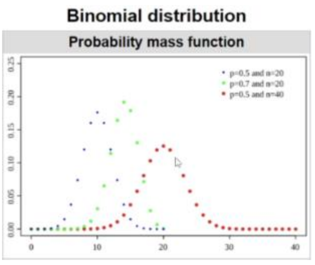
PMF : Company has launched a new Smartphone 'A'
K
(1) Use = 60% => p
(0) Not use = 40% => q = 1-p
0.6
0.4
40%
60%
0
1

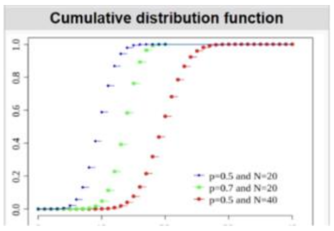
$\sigma^2 = Pr(k=0) * Pr(k=1) => p * q = 0.24$
SD = $\sqrt{pq}$

Binomial Distribution:

In probability theory and statistics, the binomial distribution with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent experiments, each asking a yes-no question and each with its own Boolean-valued outcome, success (with probability p) or failure (with probability q = 1-p). A single success/failure experiment is also called Bernoulli trial or Bernoulli experiment, and a sequence of outcomes is called a Bernoulli process, for a single trial i.e., n = 1, the binomial distribution is a Bernoulli distribution. The binomial distribution is the basis for the popular binomial test of statistical significance.

- Discrete Random Variable
- Every outcome of the experiment is binary
- These experiments are performed for n trial (Tossing a coin 10 times independently, n= 10





Notation: B(n,p)
N ∈ $\{0,1,2,3,4,\ldots.\}$ *number of trials or experiments*
P ∈ $[0,1]$ *success probability for each trail*
K ∈ $\{0,1,2,3,\ldots.n\}$ *number of successes*
q = 1-p

PMF for Binomial Distribution
Pr(K,n,P) = $C_k^n\, p^k (1-p)^{n-k}$
for K = 0,1,2,…n where $C_k^n = \frac{n!}{k!(n-k)!}$

Mean of Binomial Distribution
Mean = n*p

Variance and SD of Binomial Distribution
Variance = n*p*q
SD = $\sqrt{npq}$

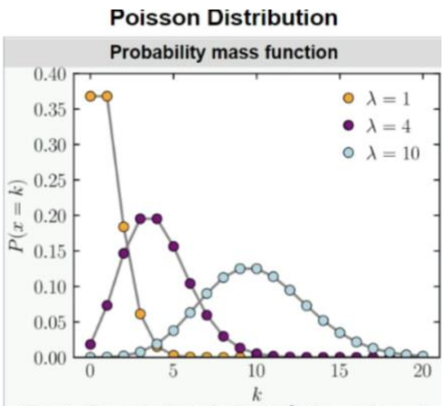
Exp: Coin flip
Number of trial (n) = 5
Probability of success (p) = 0.5
No. success (k) = varies from 0 to 5
What is the probability of getting exactly 3 heads in 5 flips
n = 5, k = 5
Pr(x=3) = $C_3^5\ (0.5)^3(1-0.5)^{5-3}$ = 0.3125

Exp: Inspecting 10 items in a factory where each item has a 10% chance of being defective. What is the probability of finding exactly 2 defective items in a sample of 10
n = 10, k = 2 p = 0.1
Pr(x=2) = $C_2^{10}\ (0.1)^2(1-0.1)^{10-2}$ = 1.937

Poisson Distribution:

In probability theory and statistics, the Poisson Distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time if these events occur with a known constant mean rate and independently of the time since the last event.

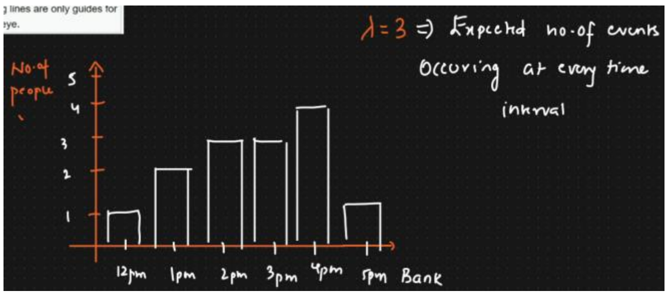
- Discrete random variable
- Describes the number of events occurring in a fixed time intervals
  exp: No. of people visiting hospital every hour
  No. of people visiting banks every hour



The horizontal axis is the index k, the number of occurrences. λ is the expected rate of occurrences. The vertical axis is the probability of k occurrences given λ. The function is defined only at integer values of k; the connecting lines are only guides for the eye.

PMF for Poisson Distribution (w.r.to pic)
P(x=5) = $\frac{e^{-\lambda}\lambda^x}{x!}$
What is the probability of person reaching at 5
P(x=5) = $\frac{e^{-3}\,3^5}{5!}$ = 0.101
What is the probability of person reaching at 4 and 5
P(x=4) + P(x=5)
What is the probability of person reaching ≤ 3
P(x=12) + P(x=1) + P(x=2) + P(x=3)

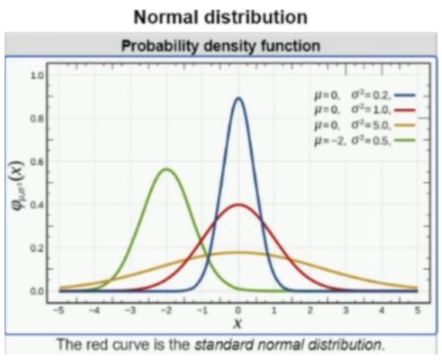
Mean of Poisson Distribution
E(x) = $\lambda * t$



Normal / Gaussian Distribution:

In probability theory and statistics, as normal distribution or Gaussian distribution is a type of continuous probability distribution for a read-valued random variable.

- Continuous random variable (PDF)
- mean = median = mode
- symmetric distribution
- σ^2 increases spread also increases
- Notation N(μ, σ^2)



The red curve is the *standard normal distribution*.

Exp: Weights of a students in a class
Heights of a students in a class

PMF for Normal / Gaussian Distribution

$$PMF = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}} (\frac{x_i - \mu}{\sigma})^2$$

Mean of Normal / Gaussian Distribution

$$\mu = \sum_{i=1}^{n} \frac{x_i}{n}$$

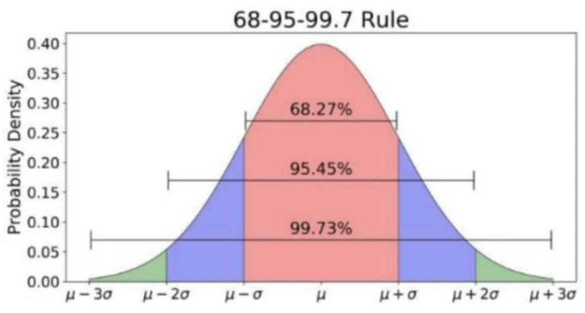
Variance and SD of Normal / Gaussian Distribution

$$\sigma^2 = \sum_{i=1}^{n} \frac{x(_i - \mu)^2}{n}$$

$$\sigma = \sqrt{variance}$$

Probability w.r.to pic (empirical rule)

Pr($\mu - \sigma \leq X \leq \mu + \sigma$) $\approx 68\%$
Pr($\mu - 2\sigma \leq X \leq \mu + 2\sigma$) $\approx 95\%$
Pr($\mu - 3\sigma \leq X \leq \mu + 3\sigma$) $\approx 99.7\%$

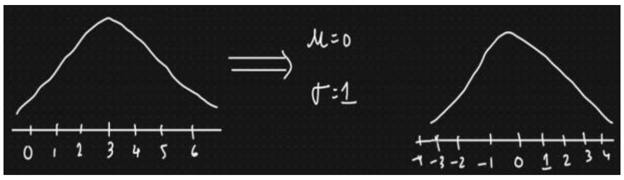


Standard Normal Distribution:

In Standard Normal Distribution, it converts the random variable into where $\mu = 0$ *and* $\sigma = 1$m by using the formula Z-score = $\frac{x_i - \mu}{\sigma}$
This helps in standardization technique where different features having different magnitude and unit to smaller and similar range in ML model training.

X = {1,2,3,4,5}

Z-score = $\frac{x_i - \mu}{\sigma}$ => y = {-2,-1,0,1,2}



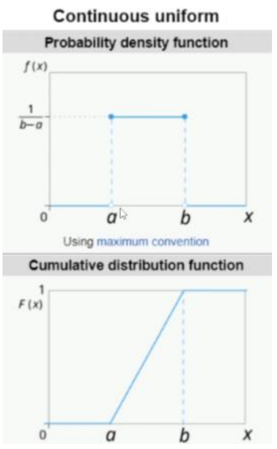
To convert the Gaussian Normal Distribution to Standard Normal Distribution use Z-score formula Z – Score = (x_i - μ) / σ
Note: For Standard Normal Distribution Mean μ is always 0 and SD σ always variance of 1

Uniform Distribution:

Continuous Uniform Distribution (continuous random variable - PDF)

In probability theory and statistics, the continuous uniform distributions or rectangular distributions are a family of symmetric probability distributions. Such a distribution describes an experiment where there is an arbitrary outcome that lies between certain bounds. The bounds are defined by the parameters a and b which are the minimum and maximum values

Notation: (a, b)
Parameters: -∞ $< a < b < \infty$



PDF = $\begin{cases} \frac{1}{b-a} & x \in [a,b] \\ 0 & otherwise \end{cases}$

CDF = $\begin{cases} 0 & for \;\; x < a \\ \frac{x-a}{b-a} & for\, x \; \in [a,b] \\ 1 & for\, x > b \end{cases}$

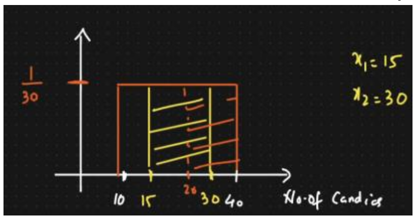
Mean = $\frac{1}{2}$ (a+b)

Median = $\frac{1}{2}$ (a+b)

Variance = $\frac{1}{2}\,(a+b)^2$

P

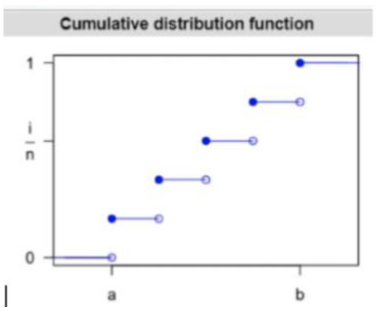
Exp: The number off candies sold daily at a shop is uniformly distributed with a maximum of 40 candies and a minimum of 10. Find the probability of daily sales to fall between 15 and 30

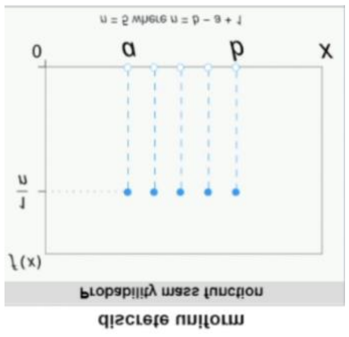


Pr(15 ≤ x ≤ 30) = $(x_2 - x_1)\frac{1}{b-a}$ = (30 -15) * $\frac{1}{30}$ = 0.5

Discrete Uniform Distribution:
In probability theory and statistics, the discrete uniform distribution is a symmetric probability distribution wherein a finite number of values are equally likely to be observed. every one of n values has equal probability 1/n. Another way of saying discrete uniform distribution would be a known finite number of outcomes equally likely to happen

- Discrete random variable
- PMF
- Exp: Rolling a dice
- Notation U(a,b)
- Parameters b ≥a
- PMF 1/n , n = b – a +1

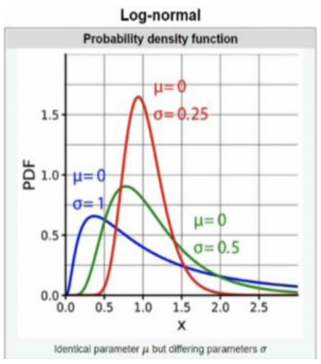




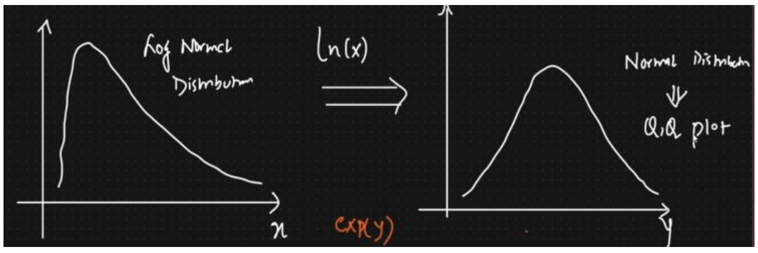
Log Normal Distribution:
In probability theory, a log-normal (or lognormal) distribution is a continuous probability distribution of a random variable whose logarithm I normally distributed. Thus, if the random variable X is log-normally distributed, then Y= ln(x) has a normal distribution. Equivalently, iff Y has a normal distribution, then the exponential function off Y.X = exp(Y) has a log-normal distribution.

X = log Normal Distribution(μ, σ)

Y ≈ $\ln(x)$ => $normal\ distribution$
X≈ $\exp(Y)$ => $\log normal\ distribution$

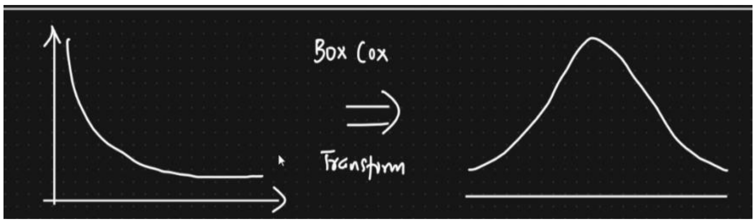


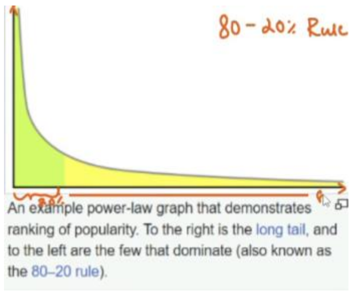
Exp: Wealth ditribution of the world



Power Law Distribution:
In statistics, a power law is a functional relationship between two quantities, where a relative change in one quantity results in a proportional relative change in the other quantity, independent of the initial size of those quantities. one quantity varies as a power of another.

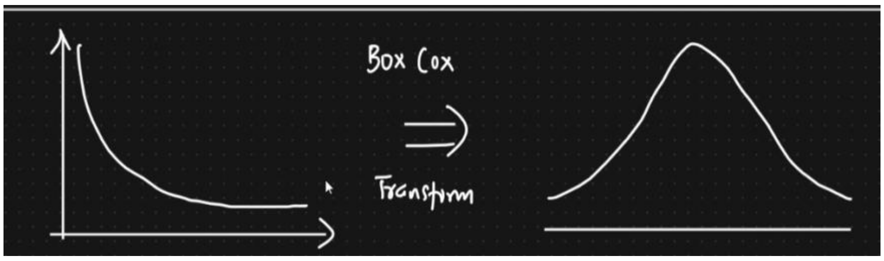
- Exp: 80% of wealth are distributed with 20% of the population
- How to convert power law distribution into normal distribution

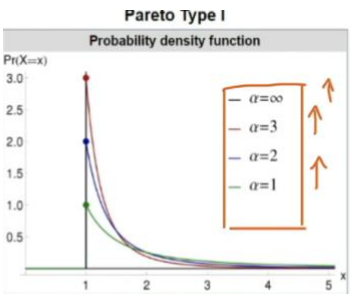




An example power-law graph that demonstrates ranking of popularity. To the right is the long tail, and to the left are the few that dominate (also known as the 80–20 rule).

Pareto Distribution:





Central Limit Theorem:

The central limit theorem relies on the concept of a sampling distribution which is the probability distribution of a statistic for a large number of samples taken from a population. The central limit theorem says that the sampling distribution of the mean will always be normally distributed, as long as the sample size is large enough. Regardless of whether the population has a normal, Poisson, Binomial or any other distribution the sampling distribution of the mean will be normal.

$$X \approx N(\mu,\sigma) \quad where\ n\ sample\ size\ is\ any\ value$$
$$X \cong N(\mu,\sigma)\ where\ n\ sample\ size\ is \geq 30$$
$$\text{sampling distribution of mean from population } X \approx N(\mu,\frac{\sigma}{\sqrt{n}})$$
$$\text{population mean } X \approx N(\mu,\sigma)$$

Estimates:

Specified observed numerical value used to estimate an unknown population parameter

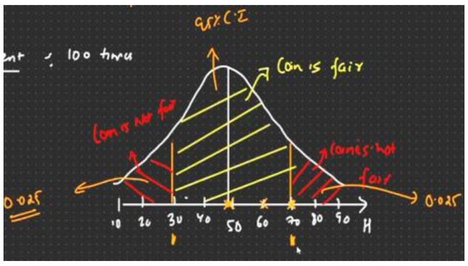
Inferential Statistics:

Hypothesis Testing and Mechanism

- Null Hypothesis H_0 (assumption to begin with)
- Alternate Hypothesis H_1
- Experiments – Statistical Analysis
- Accept the Null Hypothesis or reject Null Hypothesis

P value

The p value is a number, calculated from a statistical test, that describes how likely you are to have found a particular set of observations if the null hypothesis were true. P values are used in hypothesis testing to help decide whether to rejects the null hypothesis.



- It is is the level of marginal significance representing a given event's probability of occurrence.
- P-Value tables or spreadsheet/statistical software can be used to calculate the p-value.
- The smaller p-value indicates stronger evidence favoring the alternative hypothesis.

A p - value is a number between 0 and

- A p-value (typically ≤ 0.05) indicates strong evidence against the null hypothesis; the null hypothesis is rejected.
- A p-value (> 0.05) indicates weak evidence against the null hypothesis; the null hypothesis is not rejected.
- p-values very close to the cut-off (0.05).

Z test:

Z test for sample of $\geq 30 = Z_b = \frac{\bar{x}-\mu}{\frac{\sigma}{\sqrt{n}}}$ and Z test for population $Z_b = \frac{\bar{x}-\mu}{\sigma}$

When population standard deviation σ **is given** to perform analysis and the sample value n is ≥ 30, then use Z test

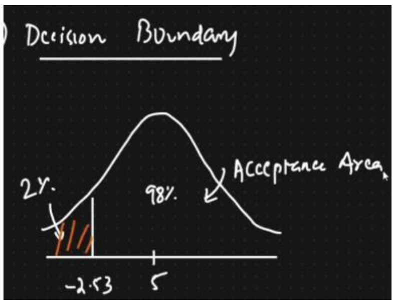
A factory manufactures bulbs with a average warranty of 5 years with standard deviation of 0.50. A worker believes that the bulb will malfunction is less than 5 years. He tests a sample of 40 bulb and find the average time to be 4.8 years.

- state null and alternate hypothesis
- at a 2% significance level, is there enough evidence to support the idea that the warranty should be revised

$\mu = 5 \; and \; \sigma = 0.50, \; n = 40$ (which is ≥ 30), $\bar{x} = 4.8$

Null hypothesis H_0 : μ = 5 years
Alternate Hypothesis H_1: $\mu < 5 \; years$ (one tail test)
Significance level = 0.02

$Z_b = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}} = \frac{4.8 - 5}{\frac{0.50}{\sqrt{40}}}$ = - 2.53



area under curve with Z score -2.53 = 0.0570 (from Z score table)
p – value = 0.0570
compare p – value with significance level: 0.0570 > 0.02, this is falling into 98% acceptance level
conclusion is the test fail to reject the null hypothesis

To convert the Gaussian Normal Distribution to Standard Normal Distribution use Z-score formula Z – Score = (x_i - μ) / σ
Note: For Standard Normal Distribution Mean μ is always 0 and SD σ always variance of 1

Student t distribution:
When population standard deviation σ is **not given** to perform analysis, then use Student t Distribution
When population standard deviation σ is given to perform analysis, then use Z test

t = $\frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}}$ where s is sample standard deviation

One sample t-test: The test will tell us whether means of the sample and the population are different

$t = \frac{\bar{x} - \mu}{s_{\bar{x}}}$ where $s_{\bar{x}} = \frac{s}{\sqrt{n}}$

Where
μ = proposed constant for the population mean
$\bar{x}$ = sample means
n = sample size (number of observations)
s = sample standard deviation
$s_{\bar{x}}$ = estimated standard error of the mean

Two sample t-test: The independent samples t test or two sample t test compares the means of two independent groups In order to determine whether there is statistical evidence that the associated population means are significantly different. The independent samples t test is a parametric test. This test is also known as independent t test.

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{s^2\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

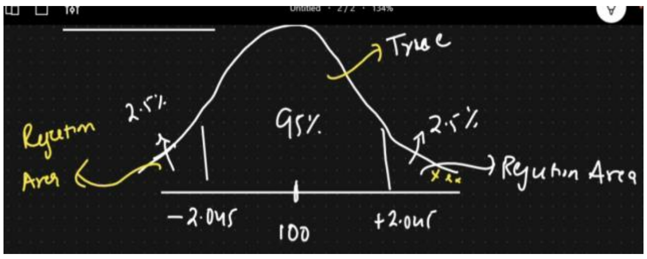
$$s^2 = \frac{\sum_{i-1}^{n_1}(x_i - x_1)^2 + \sum_{j-1}^{n_2}(x_j - x_2)^2}{n_1 + n_2 - 2}$$

In the population the average IQ is 100. A team of researches want to test a new medication to see if it has either a positive or negative effect on intelligence, or no effect at all. A sample of 30 participants who have taken the medication has a mean of 140 with a standard deviation of 20. Did the medication affect intelligence? with the confidence interval of 95%

$\mu = 100, \; n = 30, \; \bar{x} = 140, \; s = 20, \; ci = 95\%, \; \alpha = 0.05$

Null hypothesis H_0 : μ = 100
Alternate Hypothesis H_1: $\mu \neq 100 \; (two \; tail)$
Significance level = 0.05
Degree of freedom. = 30 – 1 = 29 (we need this in t table)

if t test is less than -2.045 or greater than 2.045 then reject H_0



t = $\frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}}$ = t = $\frac{140 - 100}{\frac{20}{\sqrt{30}}}$ = 10.96

since t = 10.96 which is > 2.045, then reject H_0

conclusion: the medication has increased the intelligence

When to use t test Vs Z test

- When population standard deviation σ **is given** to perform analysis and the sample value n is ≥ 30, then use Z test

- When population standard deviation σ **is given** to perform analysis and the sample value n is ≤ 30, then use t test
- When population standard deviation σ is **not given** to perform analysis, then use t test

Type 1 and Type 2 Errors:

- Reality: H_0 is True or H_0 is False
- Decision: H_0 True or H_0 is False
    - outcome 1: reject the H_0 when in reality it is false - Good
    - outcome 2: reject the H_0 when in reality it is true – Type 1 Error
    - outcome 3: accept the H_0 when in reality it is false – Type 2 Error
    - outcome 4: accept the H_0 when in reality it is true – Good

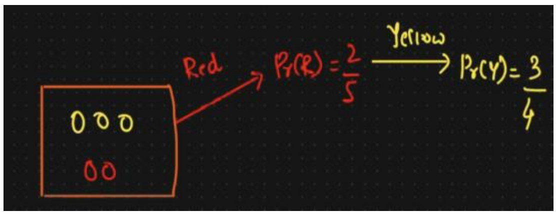
Bayes Theorem

Conditional Probability $P(B/A) = \frac{P(A \cap B)}{P(B)}$, Probability of P(B) given that P(A) is already occurred.

Independent event
Exp:
Rolling a dice Pr(1)= 1/6, Pr(2)= 1/6…
Tossing a coin Pr(H) = 0.5, Pr(T) = 0.5

Dependent event
Pr(R and.Y) = Pr(R) * Pr(Y/R)

Pr(A and B) = Pr(B and A)
Pr(A) * Pr(B/A) = Pr(B) * Pr(A/B)
Pr(B/A) = Pr(B) * Pr(A/B) / Pr(A)



Confidence Intervals and Margin of Error

confidence interval = Point Estimate $\pm$ Margin of Error

for Z test = $\bar{x} \pm Z\alpha/_2 \frac{\sigma}{\sqrt{n}}$

for t test = $\bar{x} \pm t\alpha/_2 \frac{\sigma}{\sqrt{n}}$

Chi Square Test (for categorical feature)

The test is applied when you have two categorical variables from a single population. It is used to determine whether there is a significant association between the two variables.

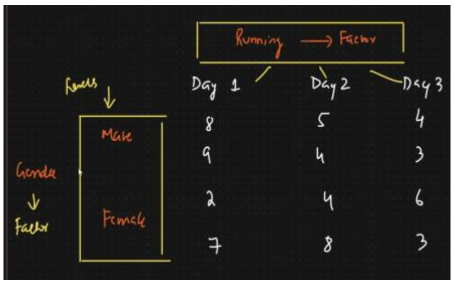
Analysis of Variance (ANOVA)

ANOVA is a statistical method used to compare the means of 2 or more groups

- Factors (variable)
- Levels
  Exp
  Medicine (factor), Dosage 5mg, 10mg, 15mg (levels)
  Mode of Pay (factor), Gpay, PhonePay (levels)

Assumption in ANOVA

- Normality of sampling distribution of mean, The distribution of sample mean is normally distributed
- Absences of Outliers, Outlier need to be removed from the dataset
- Homogeneity of Variance, Population variance in different levels of each independent variable are equal ($\sigma_1^2 = \sigma_2^2 = \sigma_3^2$)
- Samples are independent and random



Types of ANOVA

- One way ANOVA, One factor with at least 2 levels, these levels are independent
- Repeated Measures ANOVA, One factor with at least 2 levels, these levels are dependent
- Factorial ANOVA, Two or more factors each of which with at least 2 levels, levels can be independent and depende

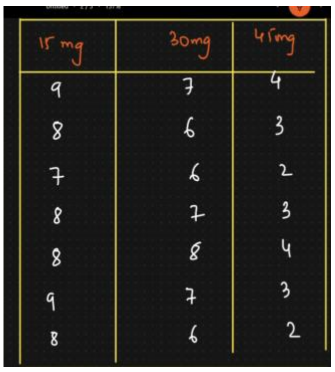
Hypothesis Testing in ANOVA (partitioning of variance in ANOVA)

Null hypothesis $H_0 : \mu_1 = \mu_2 = \mu_3 \ldots \mu_n$
Alternate hypothesis H_0 : At least one of the sample means is not equal $\mu_1 \neq \mu_2 \neq \mu_3 \ldots \neq \mu_n$

Test Statistics

F = $\frac{variance\ between\ sample}{variance\ wihin\ sample}$

Exp: One way ANOVA, Doctors want to test a new medication which reduces headache. They split the participant into 3 condition (15mg, 30mg, 45mg), Doctor ask the patient to rate the headache between (1 to 10), are there any difference between the 3 conditions using alpha 0.05



15 mg	30mg	45mg
9	7	4
8	6	3
7	6	2
8	7	3
8	8	4
9	7	3
8	6	2

One Factor (medication) and 3 level (mg)

Null hypothesis $H_0 : \mu_{15} = \mu_{30} = \mu_{45}$
Alternate hypothesis H_0 : at least one sample mean is not equal
Significant value: $\alpha = 0.05$, CI = 95%
Degree of freedom: N=21, n=7, a=3 (level)
df between = a – 1 = 3-1 = 2
df within = N – a = 21-a = 18
df total = N – 1 = 21 -1 = 20
F Test Statistics

$$F = \frac{variance\ between\ sample}{variance\ wihin\ sample}$$

Feature Engineering:

Handling Missing Values

- Mean value imputation, works with normally distributed data (**no outliers**)
- Median value imputation, works when data contains **outliers**
- Mode imputation, works well with **categorical** values
- Random sampling imputation

How to Handle Missing Values in Categorical Features?

- Remove the Rows
- Replace the Most Frequent Values (Mode)
- Apply classifier algorithm to predict
- Apply unsupervised techniques

What are the types of missing value?

1. *Missing Completely at Random MCAR*. A variable is missing completely at random (MCAR) if the probability of being missing is the same for all the observations. When data is MCAR, there is absolutely no relationship between the data missing and any other values, observed or missing, within the dataset. In other words, those missing data points are a random subset of the data. There is nothing systematic going on that makes some data more likely to be missing than other.
2. *Missing Data Not at random (MNAR)*. Systematic missing value. There is absolutely relationship between the data missing and any other values, observed or missing, within the dataset.
3. *Missing at random (MAR)*

All the techniques of handling missing value

1. Mean/ Median /Mode replacement
   Advantages:
   Easy to implement (Robust to outliers)
   Faster way to obtain the complete dataset
   Dis-Advantages:
   Change or Distortion in the original variance
   Impacts Correlation
2. Random Sample Imputation
   Random sample imputation consists of taking random observation from the dataset and we use this observation to replace the nan values. It assumes that the data are missing completely at random MCAR
   Advantages:
   Easy to implement
   There is less distortion in variance
   Dis-Advantages:
   Every situation randomness won't work
3. Capturing NAN values with a new feature
   Advantages:
   - Easy to implement
   - Captures the importance of missing values

   Dis-Advantages:
   - Creating additional Features, it may lead into curse of dimensionality
4. End of Distribution Imputation
   Advantages:
   - Easy to implement
   - Captures the importance of missingness if there is one

   Dis-Advantages:
   - Distorts the original distribution of the variables

- If missingness is not important, it may mask the predictive power of the original variable by distorting its distribution
- If the number of NA is big, it will mask true outliers in the distribution
- If the number of NA is small, the replaced NA may be considered an outlier and pre-processed in a subsequent feature

5. Arbitrary Imputation
   Advantages:
   - Easy to implement
   - Captures the importance of missingness if there is one

   Dis-Advantages:
   - Distorts the original distribution of the variables
   - If missingness is not important, it may mask the predictive power of the original variable by distorting its distribution
   - Hard to decide which value to use

6. Frequent Categories Imputation
   - Add the most frequent category to the specific feature, do it for all the feature in a dataset
   - Create new category for missing values (nan) and add to features in a dataset.
   - Use random sample to fix nan value in a dataset

   Advantages:
   - Easy to implement
   - Captures the importance of missingness if there is one

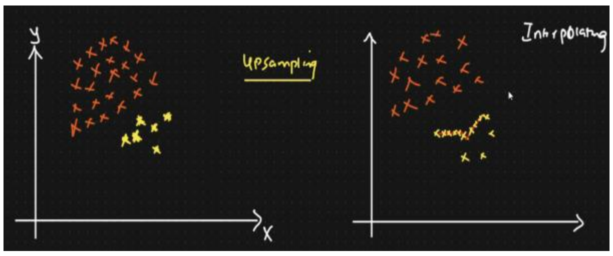
   Dis-Advantages:
   - Distorts the original distribution of the variables
   - Since we are using more frequent labels, it may use them in an over represented way, if there are more nans.

Handling Imbalanced Dataset
Under Sampling
In a data set for classification model, if the depended feature is in the ratio of 900_{YES}, 100_{No} this model will give biased result. In order to overcome this, while sampling take 100_{No} and 100_{YES} randomly is called Under Sampling (down sampling). We may not get good result if Data Set size if small, it's good to perform Under sampling technique when the Data set is big.

Over Sampling
In a data set for classification model, if the depended feature is in the ratio of 900_{YES}, 100_{No} this model will give biased result. In order to overcome this, while sampling take 100_{No} and rise this value ~ 900_{No}, so that ratio would looks like 1:1 , this technique is called Over Sampling (up sampling). In real time Over Sampling is the best solution for handling Imbalance Dataset, because in this technique

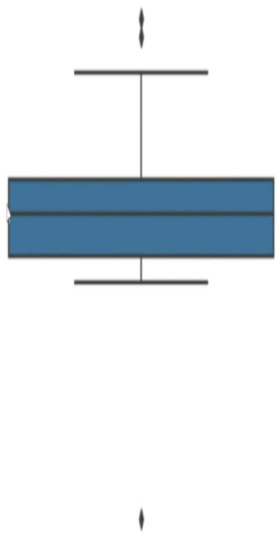
SMOTE (Synthetic Minority Oversampling Technique)

SMOTE is technique used in machine learning to address imbalanced datasets where the minority class has significantly fewer instance than the majority class SMOTE involves generating synthetic instances of the minority class by interpolating between existing instances



5 Numbers Summary and Box Plot

Min,Q1,Median,Q3,Maximum
Min,Q1,Median,Q3,Maximum np.quantile(lst_mark,[0,0.25,0.5,0.75,1.0])



IQR – Inter Quartile range
IQR = 3rd Quartile - 1st Quartile

Lower Fence = Q1 – 1.5(IQR)
Higher Fence = Q3 – 1.5(IQR)

* any value lower than lower fence or higher than higher fence is consider as outlier
* Boxplot will help to visualize outlier

Q1 = 25 percentile/100(n+1) = 25/100(20) = 5th position, which is 3
Q3 = 75/100(20) = 15th position which is 7
IQR = 7 – 3 = 4

Lower Fence = 3 – 1.5(4) = -3
Higher Fence = 7 - 1.5(4) = 13

Outliers:

- What are the criteria to identify an outlier?
  - Data point that falls outside of 1.5 times of an interquartile range above the 3rd quartile and below the 1st quartile
  - Data point that falls outside of 3 standard deviations, we can use a z score and if the z score falls outside of 2 standard deviation
- What is the reason for an outlier to exists in dataset?
  - Variability in the data
  - An experimental measurement error
- What are the impacts of having outliers in a dataset?
  - It causes various problems during our statistical analysis
  - It may cause a significant impact on the mean and the standard deviation
- Various ways of finding the outlier
  - Using scatter plots
  - Box plot
  - Using Z score $z = \frac{(X-\mu)}{\sigma}$
  - Using IQR

Outlier, Skewed and Impacts on Machine Learning Use cases

Which Machine Learning Models are Sensitive to Outliers

- Naivye Bayes Classifier – Not Sensitive to Outliers
- SVM – Not Sensitive to Outliers
- Linear Regression – Sensitive to Outliers
- Logistice Regression – Sensitive to Outliers
- Decision Tree Regressor or Classifier – Not Sensitive to Outliers
- Ensemble (RF,XGboost, GB) – Not Sensitive to Outliers
- KNN – Not Sensitive to Outliers
- Kmeans – Sensitive to Outliers
- Hierarchal – Sensitive to Outliers
- PCA – Sensitive to Outliers
- Neural Networks – Sensitive to Outliers

All Standardization and Transformation Techniques

Types of Transformations:

- Normalization and Standardization
- Scaling to Minimum and Maximum values
- Scaling to Median and Quantiles
- Gaussian Transformation
  - Logarithmic Transformation
  - Reciprocal Transformation
  - Square Root Transformation
  - Exponential Transformation
  - Box Cox Transformation

Standardization:

We try to bring all the variables or features to a similar scale. Standardization means centering the variable at zero $z = \frac{x-\bar{x}}{\sigma}$

Use Standardscalar from sklearn library

Min Max Scaling:

Min Max Scaling scales the values between 0 to 1. $X_{scaled} = \frac{X - Xmin}{Xmax - Xmin}$

from sklearn.preprocessing import MinMaxScalar

Robust Scalar:

It is used to scale the feature to median and quantiles. Scaling using median and quantiles consists of subtracting the median to all the observations, and then dividing by the interquartile difference. The interquartile difference is the difference between the 75th and 25th quantile.

IQR = 75th quantile – 25th quantile $X_{scaled} \frac{X - Xmedian}{IQR}$

from sklearn.preporcessing import RobustScalar

Gaussian Transformation:

If the dataset is not normally distribution then use Gaussian Transformation. To check whether feature is Gaussian or Normal distribution use Q-Q-Plot

- Logarithmic Transformation
- Reciprocal Transformation
- Square Root Transformation
- Exponential Transformation

- Box Cox Transformation

Standardization Vs Normalization:

- What is magnitude and Unit: (exp: age = 25 years, here 25 is magnitude and years is unit)
- Normalization (min -max Normalization) helps to scale down the feature in the dataset between 0 to 1. $X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$
- Standardization (Z – Score Normalization), Here all the features will be transformed in such a way that It will have the properties of a standard normal distribution with mean (μ) =0. And standard deviation (σ)=1. $z = \frac{(X - \mu)}{\sigma}$
- Standardization scaling mostly used technique for scaling features (Exp: Linear Regression, KNN, Clustering)
- Normalization scaling mostly used in CNN, ANN where the datasets are image and video.
- No need of scaling for AdaBoost, Random Forrest, Decision Tree, because models already splits by values.
- Scaling (Standardization or Normalization) is required when we use any ML algorithm that require gradient calculation.
- In case of neural networks, normalization is preferred since we don't assume any data distribution.
- Standardization is preferred when data follows gaussian distribution
- Standardization is preferred over normalization when there are a lot of outliers.

When should I Perform Standardization Vs Normalization

- Scaling (standardization or Normalization) is required when we use any ML algorithem that require gradient calculation (gradient boosting)
- Example of ML algorithms that require gradient calculations are linear/logistic regression and artificial neural network
- Having different scales for each feature will result in a different step size which in turn jeopardizes the process of reaching minimum point
- Scaling is not required for distance-based and tree-based algorithms such as K-Means Clustering, Support Vector Machines and K Nearest Neighbors, Decision Trees, Random Forest, and XG-Boost
- Standardization is preferred when data follows gaussian distribution
- Standardization is preferred over normalization when there are lot of outliers
- In case of neural networks, normalization is preferred since we don't assume any data distribution

Why Do we need to perform Feature Scaling?

Features in a dataset may have different Magnitude and Units of Discreet and Continuous value (exp: 10 years, 10 is magnitude, years unit), due to this we have to convert into Feature Scaling before applying any model. We have 'minmax scaling (range from -1 to 1)', 'standard scaling' techniques. We don't need Feature Scaling for RF, BT, AdaBoost

Data Encoding

- Nominal / OHE Encoding
- Label and Ordinal Encoding
- Target Guided Ordinal Encoding

Nominal / OHE Encoding

One hot encoding also known as nominal encoding, is a technique used to represent categorical data as numerical data, which is more suitable for machine learning algorithms. In this technique, each category is represented as a binary vector where each bit corresponds to a unique category. For example if we have a categorical variable 'color' with three possible values (red, green, blue) we can represent it using one hot encoding as follows red [1,0,0], green[0,1,0], blue[0,01]

Categorical features don't need to re-arrange the categories (ex;NJ,CA,WA)

- One Hot Encoding

  Covert the categories into Dummy variables by the length of total number of unique categories minus one (N unique categories -1)
- One Hot Encoding with many Categories

  Count the number of repeated categories in a feature, sort by descending order to get TOP 10 categories which are repeated the most.
  It is worth noting that the top 10 variables is a totally arbitrary number. You could also choose the top 5 or top 20.

- Mean Encoding

  It takes categories in a feature and its output value (A 1,B 2,A 0,D 5, D 2) and takes mean of output value for each category, and uses its mean value for corresponding categories. (A ->0.5, B ->2,D ->3.5)

  Disadvantage

  - Creates huge number of features, if a column has 100 categories, it creates n-1 features
  - Spars Metrix, which leads into over fitting the model while training.

Label and Ordinal Encoding

In Label encoding the categories are assigned with vales (red = 1, blue = 2, green = 3), problems with this method is when train this, model may think that red is greater than blue and vice versa.

In Ordinal encoding, it used to encode categorical data that have an intrinsic order or ranking. In this technique, each category is assigned a numerical value based on its position in the order. For example, if we have a categorical variable education level with four possible values ((high school, college, graduate, post=graduate) we can represent iiit using ordinal encoding as follows

High school 1, College 2, Graduate 3, Post-graduate 4

Categorical feature, were we re-arrange the categories based on RANK (ex: Phd, Msc, Bsc)

- Label Encoding
  Give labels in terms of Rank (ex: Phd -1 , Msc -2, Bsc -3)

Target Guided Ordinal Encoding
It is a technique used to encode categorical variables. based on their relationship with the target variable. This encoding technique is useful when we have a categorical variable with a large number of unique categories, and we want to use this variable as a feature in our marching learning model

in Target Guided Ordinal Encoding we replace each category in the categorical variable with a numerical value based on the mean or median of the target variable for that category. This creates a monotonic relationship between. the categorical variable and the target variable, which can improve the predictive power of our model

- It takes categories in a feature and its output value (A 1, B 2, A 0,D 5, D 2)) and takes mean of output value for each category and because its Ordinal Encoding at the end It Ranks the Mean (D 3.5 -> R3, B 2 ->R2, A .5 ->R2)

How to Handle Categorical Features

- One Hot Encoding
- Count or Frequency Encoding
  Advantages:
  - Easy to implement
  - Captures the importance of missingness if there is one

  Dis-Advantages:
  - It will provide same weight if the frequencies are same
- Target Guided ordinal Encoding
  Ordering the labels according to the target
  Replace the labels by the joint probability of being 1 to 0
- Mean Encoding
- Probability Ratio Encoding

EDA (Exploratory Data Analysis):
Data Cleaning process
EDA
Feature Engineering

- check data are clean
- info df.info
- describe df.describe()
- label value unique df.unique
- check for duplicate df.drop_duplicate()
- check how many categorical, numerical features in the dataset
- count how many categories in the categorical feature df.value_count(normalize=True)*100)
- correlation df.corr()
- check for balance or imbalanced data set df.value_count()
- plots: scatterplot, pair plot, catplot, heatmap, histplot, bar plot
- convert the object data type to int or float (ext 01/01/2015 to 01 01 2015 all int data type)
- convert categorical feature into integer (one hot, label, cardinal, target guided encoding
- drop feature if that is not needed for prediction

Feature Engineering

1. EDA - Exploratory Data Analysis
   Raw Data:
   1. How many Numerical Feature (Histogram, PDF), Continues, Discreate
   2. How many Categorical Feature. with cardinality (number of unique category and how many of them, rare categorical feature (category < 1% of total category of the feature)
   3. Missing Values
   4. Outliers
   5. Cleaning
   6. Relationship between dependent and independent variable
2. Handling Missing Values
3. Handling Imbalanced Data set
4. Treating outlier
5. Scaling data
6. Converting Categorical feature into numerical features

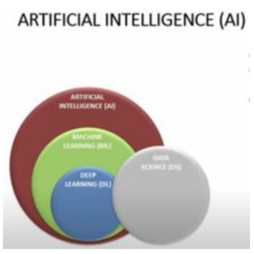
Feature Selection

1. Correlation
2. Chi square
3. Feature Importance
4. K Neighbor

Difference Between fit(), transform(), fit_transform() and predict() methods in Scikit-Learn

EDA	Feature Engineering	Feature Selection	Model Creation	Model Deployment
Data Pre-Processing Transformers * Standard Scalar – Standard Normal Distribution * MinMax Scalar * PCA – dimensionality reduction * One Hot Encoding * Remove Nulls - Imputer In this section the each data set are fit to the transformer before it get into Transformers In Sciket fit_transformers -> transformers			**Model Training** Model Selection Linear Regression Decision Tree Logistic Regression Random Forest AdaBoost Gradient Decent XgBoost In Model we do 1. **FIT and TRAIN in case of Training,** 2.Prediction with Test / New data For Training use fit_transform(x_train_data) For Test use transform(x_test_data)	

Introduction to Machine Learning



- AI is the science that enable computers to think like humans
- AI allows computers to imitate human intelligence and to thing that humans do
- AI can make decision (Exp: buy/sell stocks) , understand text (rea articles), and also see(detect faces)

- Machine Learning is a subfield of Artificial Intelligence that enables machines to improve at a given task with experience without being explicitly programmed**.**
- Note that all ML techniques are classified as AI. However, not all AI could count as ML since some basic rule-based algorithms could be classified as AI but they do not learn from experience that they do not belong to the ML category.

- Deep Learning is a subset of ML that aims at imitating the human brain using mathematical equations.
- The human brain consists of billions of neuron that communicate to each other and enable human to see, think and make decision.
- Features from input data are automatically extracted.

ML Techniques		
Supervised	Unsupervised	Reinforcement
Supervised should have dependent feature (label)		
Regression	Classification	Time Series
OLS Regression – Simple, Multiple linear regression SVM support vector machine regression Gradient Diiscent ANN Artificial Neural Network KNN K nearest neighbor Decision Tree based regression Simple Random forest Gradient Boost Ensemble Tanique Boosting Ada boosting Xg boosting Gradient boosting Regularization LASSO Ridge Elastic Net	Logistic SVM KNN ANN Decision Tree Simple Random forest Gradient Boos Ensemble Tanique Boosting Ada boosting Xg boosting Gradient boosting	
Unsupervised Grouping or Clustering, No output feature		
Dimension Reduction PCA principle compound analysis Mean MNC	Clustering K Means DB Scan	NLP CDA LSI Word2Vec
Reinforcement Model will lean by it self		

Pickling
Python pickle module is used for serializing and de-serializing python object structure. Any object in python can be pickled so that it can be save on disk. What pickle does is that it serializes the object first before writing it to file. Pickling is a way to convert a python object (list, dict, ect) into a character stream. The iidea is that this character stream contains all the information necessary to reconstruct the object in another python script

Simple Linear Regression:
Equation for straight line is y = mx + c m = slope/co-efficient c = intercept
Intercept is, at when x = 0
Slope is, a unit distance in the x axis what is the distance movement in the y axis
Prediction = $\hat{y} = mx + c$

Cost Function

cost function (mean square error) = $\frac{1}{2m}\sum_{i=1}^{m}(\hat{y} - y)^2$

m = Total Number of points

$\hat{y}$ = Line of Prediction

y = Points away from prediction line

When Cost Function is Zero (0) then, it tells all the points falls on the Regression Line

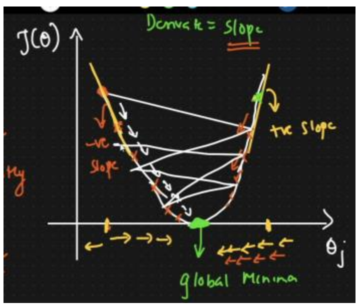
Gradient Descent
Gradient descent is an optimization algorithm used to find the value of parameters (coefficients) of a function (f) that minimizes a cost function. Common examples of algorithms with coefficients that can be optimized using gradient descent are Linear Regression and Logistic Regression.

Global Minima
In Gradient descent for optimization, to get optimal slop (coefficient) by using convergence theorem, the point it converges from -ve slop to +ve slop is called Global Minima (to obtain minimal error or best fit).

Global Minima is the point where the truth is closer to the prediction

Convergence Algorithm

$$m = m - \left(\frac{dy}{dm}\right) * L$$



L = 0.001, learning rate should control the speed off convergens, if this value is very small then converges will take long time, if this value is very high then converges will jump between +ve, -ve and miss to converge at global minima (minimum loss function / perfect predicted line)

$\left(\frac{dy}{dm}\right)$ = Slope

for -ve slope (tangent), add more +ve θ_j so that the tangent come down towards global minima

$$\begin{cases}\theta_j = \theta_j - \alpha\,(-ve)\\ \theta_j = \theta_j + \ (+ve)\end{cases}$$

* $\alpha\,(-ve)$ -ve slope, and α learning rate

for +ve slope (tangent), subtract more -ve to θ_j so that the tangent come down towards global minima

$$\begin{cases}\theta_j = \theta_j - \alpha\,(+ve)\\ \theta_j = \theta_j - \ (+ve)\end{cases}$$

Multiple Linear Regression
Multiple Linear Regression has multiple more than one independent feature and one dependent feature

$h_0(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$, where $\theta_1\theta_2\theta_3$ are coefficient and θ_0 is intercept

Performance Metrics

R^2 give how the model is performing, more towards 1 is considered good performance

$R^2 = 1 - \frac{SSres}{SSmean}$ SSres = $\frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$ SSmean = $\frac{1}{n}\sum_{i=1}^{n}(\bar{y} - y)^2$

R^2 gives goodness of best fit in regression model, with range from 0 to 1, and closer to 1 is best fit.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y - \hat{y})^2}{\sum_{i=1}^{n}(y - \bar{y})^2}$$

$SS_{res} = Sum\ of\ recidual\ or\ error$, $SS_{tot} = Sum\ of\ average\ total$

$\hat{y} = Predictted\ \widehat{Regression}\ Line,\ \ \bar{y} = Average\ of\ Total\ data\ points$

SSmean (MSE) Mean Squared Error is to calculated the difference between each target y and the model's predicted value $\bar{y}$ (i.e., the residual)

***Adjusted R^2*,** helps to overcome the issue with R^2 value, i.e., when a feature that are not correlated with the label feature, still the R^2 value will increase considerably when compare to the features that are correlated with label feature, with Adjusted R^2 it penalizes with respect to the feature that are not correlated with the output feature.

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(N-1)}{N-p-1}$$

R^2 = sample R square, p = Number of predictors (independent features), N = Total sample size

Why $R^2_{adjusted}$ is used, when we use multiple features ($y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$) the co-efficient of each feature will tend to increase R^2 value, including the feature which does not corelate with output, which will lead into wrong prediction. To avoid such errors, we use $R^2_{adjusted}$.

When number of features increases and corelated with output then the $R^2_{adjusted}$ value will increase little than R^2,but when features are not corelated with output then the $R^2_{adjusted}$ will degrees than R^2

Difference of $\mathbf{R^2}$ and $\mathbf{R^2_{adjusted}}$

- Every time you add an independent variable to a model, then R^2 increases, even if the independent variable is insignificant, it never declines. Whereas $R^2_{adjusted}$ increases only when independent variable is significant and affects dependent variable.
- $R^2_{adjusted}$ value always be less than or equal to R^2 value.

Sagemaker

- One limitation of R^2 is that it increases by adding independent variables to the model which is misleading since some added variables might be useless with minimal significance.
- Adjusted R^2 overcomes this issue by adding penalty if we make an attempt to add independent variable that does not improve the model
- Adjusted R^2 is a modified version of the R^2 and takes into account the number of predictors in the model
- If useless predictors are added to the model, Adjusted R^2 will decrease
- If useful predictors are added to the model, Adjusted R^2 will increase
- K is the number of independent variables and n is the number of samples $R^2{}_{adjusted} = 1 - (\frac{(1-R^2)(n-1)}{n-k-1})$

MSE,MAE,RMSE

Mean Square Error (MSE) = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Advantage
- Differentiable at any points while convergence
- It has local and global minima
- Converge fast

Disadvantage
- Not robust to outlier, MSE will increase
- No longer with same unit, due to square

Mean Absolute Error (MAE) = $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

Advantage
- Robust to outlier, due to not square (MAE will increase but not compare to MSE)
- It will be in same unit, due to not square

Disadvantage
- Convergence take more time
- Time consuming

Root Mean Square Error (RMSE) = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

represents the standard deviation of the residual (ie., differences between the model prediction and the true values (training data))

Advantage
- Differentiable
- It will be in same unit, due to not square

Disadvantage
- Robust to outlier

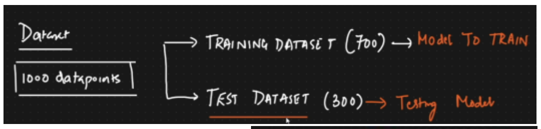
Least Sum of Square (LSS) = min $\sum(\hat{y}_i - y_i)^2$

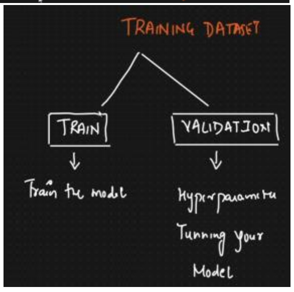
Mean Absolute Error Percentage (MAPE) = $\frac{100\%}{n}\sum_{i=1}^{n}|(y_i - \hat{y}_i)/y_i|$

Mean Percentage Error (MPE) = $\frac{100\%}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)/y_i$

Overfitting and Underfitting (Bias and Variance)

Underfitting In Training data set, the model gives more error, then the model is called Underfitting.
In Underfitting, the accuracy will be low for both Training and Test data set, and because of more error It has High Bias for Training data set, High variance for Test data set.
High Bias, High Variance


Dataset
1000 datapoints
TRAINING DATASET (700) → Model To TRAIN
TEST DATASET (300) → Testing Model



Overfitting In Training data set, all the data are covered by the model, then the model is called Overfitting. This kind of model will give more error (variance) for Test data set. In Overfitting, accuracy will be high for Training data but accuracy will be low for Test data. In Overfitting, because of error is low for Training data set, it has Low Bias for Training data set, and High Variance for Test data set.
Low Bias, High Variance

Note: Bias meaning error of Training dataset, Variance meaning error of Test dataset

Binary Tree Classifier is example for Over Fitting, Random Forest Classifier is example for Low Bias, Low Variance.

- Best model will have Low Bias and Low Variance
- When our error is biased, it means the model's prediction is consistently far away from the actual value
- This could be a sign of poor sampling and poor data
- One objective of a biased model is to trade bias error for generalized error. We prefer the error to be more evenly distributed across the model.

Underfitting	High Bias – High Error – Training Data	**Overfitting**	Low Bias – Low Error. – Training Data	**Best fitting**	Low Bias
	High Variance – High Error. – Test Data		High Variance – High Error. – Test Data		Low Variance

Linear Regression using OLS (ordinary least square)
Cross Validation
After train a model with Training dataset, we perform accuracy of the model by using Test dataset. The accuracy may change due to Train-Test dataset split ratio (70:30) and while changing number of random sample test.
Cross Validation allows us to compare different machine learning models and get a sense of how well they will work in practice OR is to test the model's ability to predict new data

Types of Cross Validation
1. Leave One Out CV (LOCV)
   - Say out of 1000 records, it takes ONE dataset as Test and Remaining as Train, this task will happen for 1000 records
   - Generate several models on different cross sections of the data
   - Measure the performance of each
   - Take the mean performance
   - Disadvantage – Needs high processing power due to Train-Test split for all records, leads into low bias (overfitting)
2. K Fold CV
   - Split the data into K group
   - Train the model on all segments except one
   - Test model performance on the remaining set
   - If K=5, split the data into five segments and generate five models
   - Disadvantage – unbalanced dataset gives more biased output
3. Stratified CV
   - Overcome the problem from K – Fold CV
   - It makes sure the number of Yes and No are equal ration in both Test-Train
4. Time Series CV

Multicollinearity
In a regression model if features are correlated (>90) each other then it is called Multicollinearity.
In this type of model, we can take one feature among the corelated the features, an can drop the remaining.
$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$ If features x_1, x_2 are highly correlated (>90) then feature x_1 or x_2 can take for regression model.

SageMaker Linear Learner:
Preprocessing:
- Normalization or feature scaling is offered by Linear Learner
- Feature scaling is a critical preprocessing step to ensure that the model does not become dominated by the weight of a single feature

Training:
- Linear Learner uses stochastic gradient descent to perform the training
- Select an appropriate optimization algorithm such as Adam, AdaGrad
- Hyperparameters can be selected and turned (exp: learning rate)
- Overcome model overfitting using L1, L2 regularization

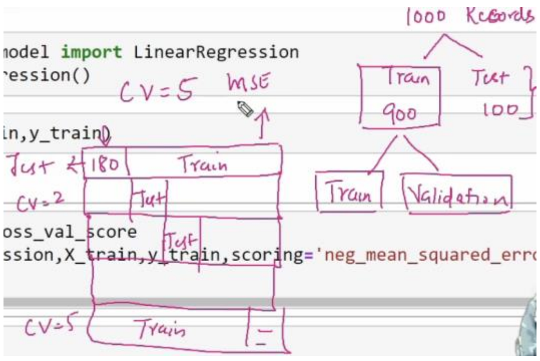
Validation:
Trained models are evaluated against a validation dataset and best model selected baased on the following metrics:
- For regression: mean square error, root mean square error, absolute error
- For classification: F1 score, precision, recall, or accuracy

Hyperparameter Optimization for Xgboost
Hyperparameter Optimization is very important task for any ML techniques, reason is it helps the model to use correct parameter. How Xgboost Hyperparameters are selected / optimized by using Hyperparameter Optimizer like Radom Search, Grid Search, Bayesian Optimizer

Simple Linear Regression

```
# to read file
df = df.read_csv(file.csv)

# select independent and dependent feature
X = df[['weight']]  # independent features should be a data frame or 2 dimensional array
y =  df['Height']      # this variable can be in series or 1 dimension array
X_serias = df['weight']
np.array(S_series).saphe => (23,)
X_serias = df[['weight']]
np.array(S_series).saphe => (23,1)

# to select train, test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 42)

# standardization
from sklean.prepocessing import StandardScaler
scalar   = StandardScalar()
X_train = scalar.fit_transform(X_train)
X_test  = scalar.transform(x_test)  *avoid data leakage (fit_transform and transform)

# model selection
from sklearn.liner_model import LinearRegression
regression = LinearRegression(n_jobs.= -1)
regression.fit(X_train,y_train)
print(f'Coeeficiient or slope',regression.coef_)
prinnt(f'intercept',regression.intercept_)

# prediction for test data
y_pred = regression.predict(x_test)

# performance metric
from sklearn.metris import mean_absolute_error,mean_squared_error,r2_score
mse  = mean_squared_error(y_test,y_pred)
mae  = mean_absolute_error(y_test,y_pred)
rmse = np.sqrt(mse)
r2_score = r2_socre(y_test,y_pred)
adj_r2 = 1 -(1-r2_score)(len(y_test)-1)/(len(y_test) -x_test.shape[1]-1)
print(mse,mae,rmse,2_score)

# prediction for new data
regression.predict(scalar.transform([[72]])) *scale the data to be predicted before prediction
```

Multiple Leaner Regression



```
model import LinearRegression
ression()
in,y_train)
oss_val_score
ssion,X_train,y_train,scoring='neg_mean_squared_erro
```

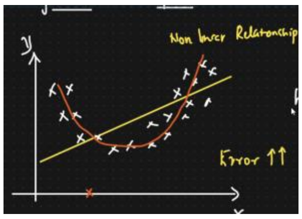
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# to read file
df = pd.read_csv(file.csv)

# drop unnecessary columns
df.drop(column=['come_column'],axis = 1,inplace = True)
df.corr()
plt.scatter(df[a],d[b],color='r')

# select independent, dependent features
X = df.iloc[:,:-1]
y = df.iloc[:,-1]

# train, test split
from sklearn.model_selection import train_test_split
X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,Random_size = 42)

# standardization
from sklearn.preprocessing import StandardScaler
Scalar  = StandardScaler()
X_train = Scalar.fit_transform(X_train)
x_test   = Scalar.transform(x_test) * we don't have to do scaling it for y or label feature

# model selection
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train,y_train)
```

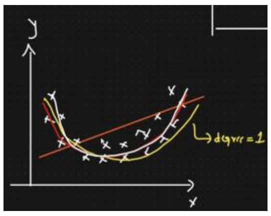
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html
https://scikit-learn.org/1.5/modules/model_evaluation.html

```
#cross validation
from sklearn.model_selection import cross_val_score
validation_score = cross_val_score(regression,X_train,y_train,scoring = 'neg_mean_squared_error',CV = 5)
                              * by using  this CV the result will give 5 different value of neg_mean_squared_error

validation_score
np.mean(validation_score)  *to get mean of 5 mse

#prediction
y_pred = regression.predict(x_test)
```

```
y_pred

#performance Metrics
from sklearn.metris import mean_absolute_error,mean_squared_error,r2_score
mse  = mean_squared_error(y_test,y_pred)
mae  = mean_absolute_error(y_test,y_pred)
rmse = np.sqrt(mse)
r2_score = r2_socre(y_test,y_pred)
adj_r2 = 1 -(1-r2_score)(len(y_test)-1)/(len(y_test) -x_test.shape[1]-1)
print(mse,mae,rmse,2_score)

#assumptions
plt.scatter(y_test,y_pred)
residuals = y_test - y_pred
sns.displot(residuals,kind=kde)
plt.scatter(y_pred,esiduals) * shouldn't have any pattern
```

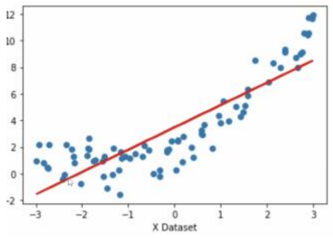
Polynomial Regression

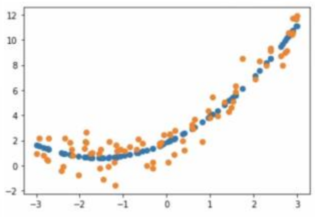




$h_\theta(x) = \beta_0 + \beta_1 x$ simple linear regression
$h_\theta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$ multiple linear regression

when polynomial degree is 0 $h_\theta(x) = \beta_0 x^0$, constant
when polynomial degree is 1, $h_\theta(x) = \beta_0 x^0 + \beta_1 x^1$, simple linear regression
when polynomial degree is 2, $h_\theta(x) = \beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2$
when polynomial degree is n, , $h_\theta(x) = \beta_0 x^0 + \beta_1 x^1 + \beta_2 x^2 \ldots \ldots + \beta_n x^n$





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

#generate data to mimic polynomial
X = 6 * np.random.rand(100,1) -3
y = 0.5 * X**2 + 1.5*X +2 +np.random.radn(100,1)

plt.scatter(X,y,color='g')

from sklearn.model_selection import train_test_split
X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state =42)

from sklearn.linear_model import LinearRegression
regression = LinearRegression

regression.fit(X_train,y_train,)

from sklearn.metrics import r2_score
score = r2_score(y_test,regession.predict(x_test))

#apply polynomial transformation
from sklean.preprocessing import PolynomialFeature

poly = PolynomialFeature(degree=2,include_biase=True)
X_train_poly = poly.fit_tranform(X_train)
x_test_poly = poly.transform(x_test)

#model selection
regession = LinearRegression
regession.fit(X_train_poly,y_train)
y_pred = regressiion.predict(x_test_poly)
score = r2_score(y_test,y_pred)

plt.scatter(x_train,x_train_poly)
plt.scatter(x_train,y_train)

#prediction of new data
X-new_poly = poly.transform(x_new)

# using pipeline
from sklearn.pipeline import Pipeline

def poly__regression(degree):
X_new = np.linespace(-3,3,200).reshape(200,1)

poly_features = PolynomialFeatures(degree=degree,include_bias=True)
lin_reg = LinearRegression()
poly_regression=Pipeline([('poly_features',poly_features),('lin_reg',lin_reg)])

poly_regression.fit(X_train,y_train). * polynomial features, then fit liner regression
y_pred_new= poly_regression.predict(X_new)
```

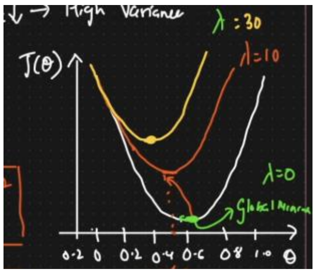
Ridge, Lasso, Elastic Net Regression (Hyperparameter tuning the linear regression)
Ridge and Lasso regression helps the models to reduce overfitting (Low Bias, High Variance) model to Low Variance, Low Bias

- In Ridge regression the slope value will tend to move towards to zero (0) value.
- In Ridge regression it helps to reduce the overfitting model as well as for multiple feature analysis. In Lasso regression helps for feature selection, when slope value is closer to zero (0) we can drop this feature and use rest of the feature for further analysis
- Lasso regression (L1 regression) can be useful if we have several independent variables that are useless

- Ridge regression can reduce the slope close to zero (but not exactly zero) but Lasso regression can reduce the slope to be exactly equal to zero.
- **When to choose L1:** If you believe that some features are not important and you can afford to lose them, then L1 regularization is a good choice. The output might become sparse since some features might have been removed.
- **When to choose L2:** If you believe that all features are important and you'd like to keep them but weigh them accordingly.

Ridge Regression = $\sum_{i=1}^{m}(\hat{y}-y)^2 + \lambda * (slope)^2 \quad \lambda - may\ be\ any + ve\ value$
Lasso Regression = $\sum_{i=1}^{m}(\hat{y}-y)^2 + \lambda * |\ slope\ |$

Ridge Regression (L2 Regularization or Hyperparameter tuned regression)

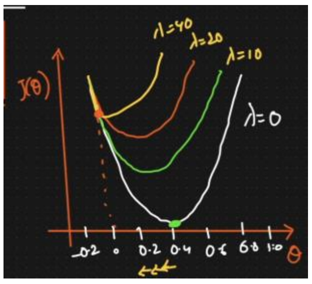


Ridge regression used to solve Overfitting issue, that causes Low Bias (training data), High Variance (test data)

Ridge Regression = $\sum_{i=1}^{m}(\hat{y}-y)^2 + \lambda * (slope)^2 \quad \lambda - may\ be\ any + ve\ value,$
$and\ \lambda\ is\ inversely\ propostional\ to\ slope$

when cost function (mean square error) = $\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}-y)^2$ = 0, then, it tends to low bias and overfit while training data, in order to avoid overfitting, add λ which is small variance.

Lasso Regression (L1 Regularization)



Lasso regression used for feature selection, by adding λ value to the slope, it tends to make the slope(coefficient) to 0 and avoid that feature for model training.
Lasso Regression = $\sum_{i=1}^{m}(\hat{y}-y)^2 + \lambda * |\ slope\ |$

Elastic Net Regression
Elastic Net Regression is a combination of Ridge, Lasso Regression

Elastic Net = $\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}-y)^2 + \lambda * (slope)^2 + \lambda * |\ slope\ |$

Types of Cross Validation
- Leave One Out CV (LOCV)
  - Say out of 1000 records, it takes ONE dataset as Test and Remaining as Train, this task will happen for 1000 records
  - Generate several models on different cross sections of the data
  - Measure the performance of each
  - Take the mean performance
  - Disadvantage – Needs high processing power due to Train-Test split for all records, leads into low bias (overfitting)
- K Fold CV
  - Split the data into K group
  - Train the model on all segments except one
  - Test model performance on the remaining set
  - If K=5, split the data into five segments and generate five models
  - Disadvantage – unbalanced dataset gives more biased output
- Stratified CV
  - Overcome the problem from K – Fold CV
  - It makes sure the number of Yes and No (binary date) are equal ration in both Test-Train
- Time Series CV

Performance Metrics for Classification Problem in ML
In classification problem especially for Binary classification if the data is unbalanced (ie., 80% - 20% or 90% - 10%) the ML model for Binary classification will be biased on the higher percentage. To come out of this problem we use Confusion Matrix

		Actual Value	
Predicted		1	0
	1	TP	FP
	0	FN	TN

FP = False Positive (FPR, Type 1 Error, Specificity)
FPR Ratio = $\frac{FP}{(FP+TN)}$
FN = False Negative (FNR, Type 2 Error)
Accuracy = $\frac{TP+TN}{(TP+FP+FN+TN)}$
Misclassification rate (Error Rate) = $\frac{FP+FN}{(TP+FP+FN+TN)}$

$Recall = \frac{TP}{TP+FN}$ Recall (TPR, Sensitivity) – Out of all the actual +ve value, how many +ve value predicted correctly (When the class was actually TRUE, how often did the classifier get it right). Whenever FN is much important then use Recall.

$Precission = \frac{TP}{TP+FP}$ Precision (+ve Predicted Value) – Out of all the predicted +ve value, how many values are correctly predicted +ve (When the model predicted TRUE class, how often was it right) . Whenever FP is much important then use Precision

F βeta Score: $F_\beta = (1+\beta^2)\frac{Precision*Recall}{\beta^2*Precision+Recall}$

We have to choose F_β score when FP and FN are more important, in this we have to choose the value of β. The value of β may change depends on FP and FN.

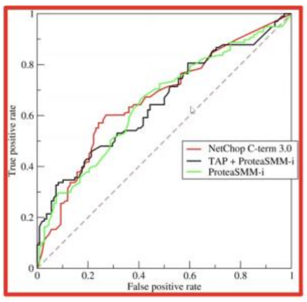
Exp:In confusion matrix if both FP and FN are equally important then select β =1.
when β =1 then $F_\beta = (2)\frac{Precision*Recall}{\beta^2*Precision+Recall}$ Which gives Harmonic Mean = 2 $\frac{x*y}{x+y}$.

Exp: In confusion matrix if FP (Type one Error) is more important than FN (Type two error) then select β = 0.5
when β =0.5 then $F_\beta = (1+0.25)\frac{Precision*Recall}{0.5^2*Precision+Recall}$

Exp. In confusion matrix if FN (Type two error) is more important than FN (Type one error) then increase the β value ie., β=2
when β =2 then $F_\beta = (1+4)\frac{Precision*Recall}{4^2*Precision+Recall}$

F_β Score Summary:
$F_{1\,Score}$ $is,$ β =1 for FP (Type one error, Precision), FN (Type two error, Recall) has higher impact.
$F_{0.5\,Score}$ $is,$β =0.5 for FP (Type one error, Precision) has higher impact than FN (Type two error, Recall)
$F_{2\,Score}$ $is,$ β =2 for FN (Type two error, Recall) has higher impact than FP (Type one error, Precision)
When FP & FN have greater impact then select β =1
When FP has more impact then reduceβ =0.5
When FN has more important than increase β > 2

ROC (Receiver Operation Characteristic) – ROC curve is created by plotting TP against FP at various models' threshold



The true-positive rate is also known as sensitivity, recall or probability of detection in machine learning.

The false-positive rate is also known as the probability of false alarm and can be calculated as (1-specificity)

ROC Helps to determine best threshold value
AUC (Accuracy Under the Curve) – Summarizes the impact of TPR and FPR is a single value
AUC Helps to determine which classification model is best

Parameter:

- Values that are obtained by the training process such as network weights and biases
  exp: In Linear Regression y=mx +b, in this, model is trying to get the value of m, b by training

Hyperparameter:

- Values set prior to the training process such as number of neurons, layers, learning rate etc.
  exp: In regression model setting up 'learning rate' to reach 'Global Minimum' (least error)

Batch Size

- Batch size indicates the number of samples that will propagate through the algorithm
  Exp: Let's assume that we have 1000 images for training. For batch size = 50, the first 50 images (from index 1 to index 50) will be propagated to the training algorithm and used for training. Then the next 50 images are propagated (index 51 to index 100). Procedure is repeated until we use all the training data.
- If the batch size is small, ML models can easily escape local minimum areas
- If the batch size is large, ML model can get stuck in a local minimum.

Hyperparameters Optimization:
- ***Grid Search***

- GridSearch preforms exhaustive search over a specified list of parameters
  Note that you will have the following number of combinations 3*3*3*2=54

We will run each combination 5 times since we set the cross validation =5
Total number of runs = 54*5 = 270

```
parameter_grid = {'max_depth':[3,6,10],
                  'learning_rate':[0.01,0.05,0.1],
                  'n_estimators':[100,500,1000],
                  'colsample_bytree':[0.3,0.7])
grid = GridSearchCV(estimator = model,
                    param_grid = 'neg_mean_squared_error',
                    cv = 5,
                    verbose = 5)
```

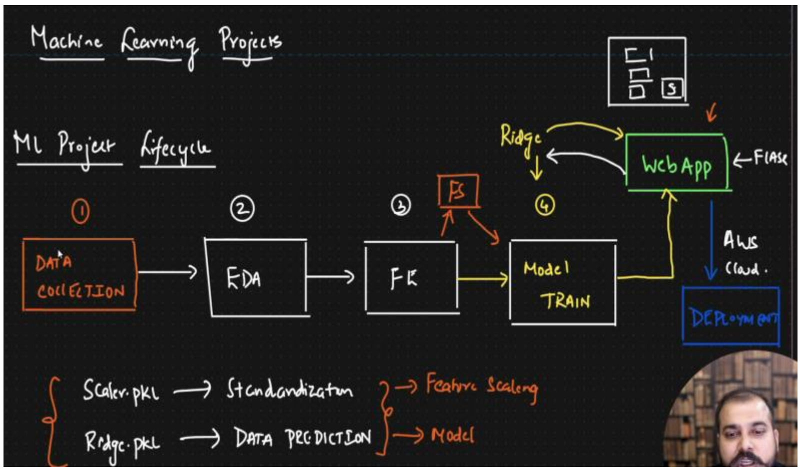
- ***Randomized Search***
  - Grid search works great if the number of combinations are limited
  - In scenarios when the serach space is large, RandomizedSearchCV is preferred.
  - The algorithm works by evaluating a select few numbers of random combinations

```
grid = { 'n_esstimaters': [100,500,900,1100,1500]
         'max_depth': [2,3,5,10,15]
         'learning rate':[0.05,0.1,0.15,0.20]
         'min_child_weight':[1,2,3,4]
         'booster':['gbtree','gblinear'])
random_cv = RandomizedSearchCV(estimator=model,
                               param_distribution = grid,
                               cv =5,
                               n_item = 50,
                               scoring = 'neg_mean_obsolute_error',
                               verbose = 5,
                               return_train_score = True)
```

- ***Bayesian Optimization***
  - Bayesian optimization overcomes the drawbacks of random search algorithms by exploring search spaces in a more efficient manner
  - If aa region in the search space appears to be promising (ie., resulted ina small error), this region should be explored more which increases the chances of achieving better performance.

```
search_space = ('max_depth':[4,20]
                'n_estimators':[100,500],
                'learning_rate':[0.01,1.0,'log-uniform'])
xgb_bayes_search = BayesSearchCV(model,
                                 search_space,
                                 n_iter =50,
                                 scoring = 'neg_mean_absolute_error',
                                 cv=5)
xgb_bayes_search.fit(X_train,y_train)
```

Cleaning of Data Set


Machine Learning Projects
ML Project Lifecycle
① DATA COLLECTION
② EDA
③ FE
FS
④ Model TRAIN
Ridge
Web App
FlASK
AWS Cloud.
DEPLOYMENT
Scaler.pkl → Standardization → Feature Scaling
Ridge.pkl → DATA PREDICTION → Model

Logistic Regression (Binary Classification)

- Usually Logistic Regression is mostly used for Binary classification, with some Hyper tuning it can be used for Multiclass classification.
- In Binary classification, the Logistic Regression is used with the intuition that it can linearly separate this two classification groups.
- In Logistic regression, Cost function should be always maximum ie., $Max\ \sum_{i=1}^{n} y_i * w^t x_i$ where w^t is co-efficient of feature x_i. $w^t x_i$ which will give the distance of y_i from the plane or linear straight line.
- In logistics regression for binary classification, when there is an outlier it tent to increase the value (ie., $\sum_{i=1}^{n} y_i * w^t x_i$), which is wrong, to avoid this we use sigmoid function $\frac{1}{1+e^z}$ where z is $\sum_{i=1}^{n} y_i * w^t x_i$ The sigmoid function range from 0 to 1.
- Regression – output contains continuous numeric values

- Binary classification – output label must be either 0 or 1
- Multiclass classification – output label must be from 0 to num_class -1

Logistic Regression Multiclass classification (One Vs Rest)

We can use Logistic regression for Multiclass classification by making one feature with +ve label and rest of the features as -ve label.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.dataset import make_classification
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

from sklean.linear_model import logisticRegression

logistic = logisticRegression()
logistic.fit(X_train,y_train)

y_predd = logistic.predict(X_test)

from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

score = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
cls = classification_report(y_test,y_pre)
print(score,cm,cls)
```

Hyperparameter tuning - Grid Search

```
model = LogisticRegession()
penalty=['l1','l2','elasticnet']  * parameters from LogisticRegession()
c_value=[100,10,1.0,0.1,0.01] * parameters from LogisticRegession()
solver=['newton-cg','lbfgs','liblinear','saag','saga']. * parameters from LogisticRegession()
params = dic(panalty=penalty,C=c_value,solver = solver)

from sklean.model_selection import GridSearhCV
from sklean.model_selection import StratifiedFiled

cv = StratifiedFiled()
grid = GridSearhCV(estimator = model,param_grid=params,cv=cv,n_jobs=true)
grid.fit(X_train,y_train)

y_pred = grid.predict(x_test)

grid.best_params_
grid.best_score_
score = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
cls = classification_report(y_test,y_pre)
```

Logistic OVR

```
from sklean.linear_model import logisticRegression
logistic = logisticRegression(multi_class='ovr')
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

logistic = logisticRegression()
logistic.fit(X_train,y_train)

y_predd = logistic.predict(X_test)

from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

score = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
cls = classification_report(y_test,y_pre)
print(score,cm,cls)
```

Logisticregression Imbalance data set

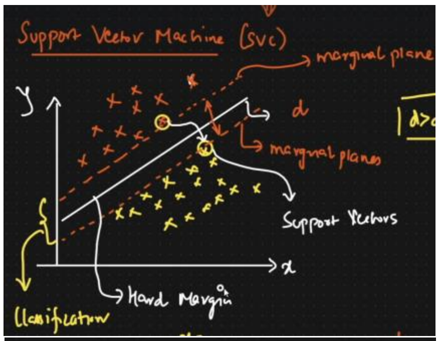
```
from sklean.linear_model import logisticRegression
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

logistic = logisticRegression()
penalty=['l1','l2','elasticnet'] * hyperparameter tuning
c_value=[100,10,1.0,0.1,0.01]  * hyperparameter tuning
solver=['newton-cg','lbfgs','liblinear','saag','saga'] * hyperparameter tuning
class_weight = [0:w,1:y for w in [1,10,50,100] for y in [1,10,50,100]]  * hyperparameter tuning

params = dic(panalty=penalty,C=c_value,solver=solver,class_weiight=class_weight)

from sklean.model_selection import GridSearhCV
from sklean.model_selection import StratifiedFiled
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

cv = StratifiedFiled()
grid = GridSearhCV(estimator = model,param_grid=params,cv=cv,n_jobs=true)
grid.fit(X_train,y_train)

y_pred = grid.predict(x_test)

grid.best_params_
grid.best_score_
score = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test,y_pred)
cls = classification_report(y_test,y_pre)
```

Logistic Regression with ROC curve and ROC AUC score

```
from sklean.linear_model import logisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=42)

model = LogisticRegression()
model.fit(X_train,y_train)
model_prob = model.predict_proba(X_test)

dummy_model_auc = roc_auc_score(y_test,model_prob)
model_auc = auc_score(y_test,model_prob)
print(dummy_model_auc,model_auc)
```

Support Vector Machine (SVM) Classifier (output value is binary)
SVM helps to solve both Regression and Classification problems

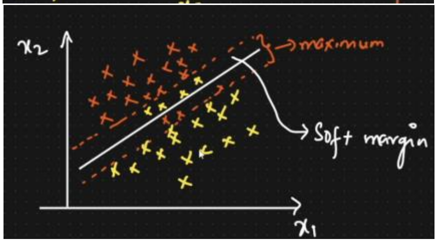
Support Vectors– +ve or -ve points that helps to create marginal planes parallel to Hyperplanes are called Support Vectors



Hyperplanes – Hyperplane is a regression line/plane that separates the groups (+ve, -ve) of datasets in classification problem

Marginal Distance – SVM makes sure to create two more hyperplanes that is parallel to Hyperplane, and also it passes through the nearest points of +ve, and -ve from Hyperplane. The distance between Hyperplane to +ve, -ve Hyperplane is called Marginal Distance.

Idea of creating Hyperplane is to find plane that separates the +ve/-ve point with maximum distance. We can create more than one Hyperplane for a single dataset, but the best Hyperplane will have maximum Marginal Distance.



Linear Separable – dataset in classification problem, +ve or -ve groups are separated linearly with Hyperplane (simple straight line)

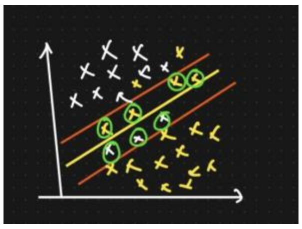
Non-Linear Separable - dataset in classification problem, +ve or -ve groups are not separated linearly is called Non-Linear Separable, but this kind of problem can be solved by SVM Kernel.

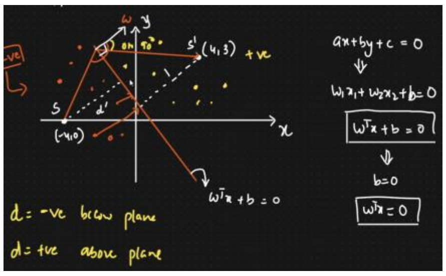
SVM Kernel – SVM Kernel is used to separate the +ve and -ve groups that are non-linear, by converting (or creating another dimension) 2D to 3D space, which helps to create Hyperplane between +ve, -ve groups.

Soft Margin, Hard Margin – In soft margin data point may overlap each other due to this error will occur, In Hard Margin there is not overlap in the data set.

Cost Function related to soft margin = $\frac{\|w\|}{2} + c_i \sum_{i=1}^{n} \xi_i$

*c_i *is hypyer parameter, for how many points we want to avoid misclassification*
∗ ξ_i summation of the distance of the incorrect data points from the marginal plane

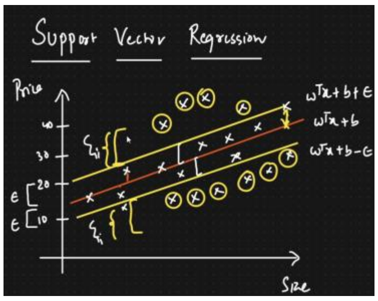




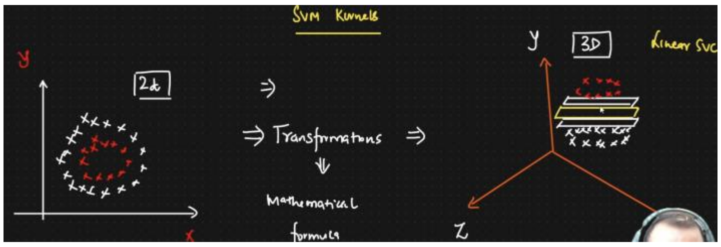
Support Vector Machine (SVM) Regression (output value is continuous)

Cost Function = $\frac{\|w\|}{2} + c_i \sum_{i=1}^{n} \xi_i$
*c_i *is hypyer parameter, for how many points we want to avoid misclassification*
∗ ξ_i summation of the distance of the incorrect data points from the marginal plane

Constrain = $|y_i - w_i x_i| \leq \in + \xi$
∗ ∈ marginal error
∗ ξ error above the margin



SVM Kernel – SVM Kernel is used to separate the +ve and -ve groups that are non-linear, by converting (or creating another dimension) 2D to 3D space, which helps to create Hyperplane between +ve, -ve groups
Types of SVM Kernel
- Polynomial Kernel;
- RBF Kernel
- Sigmoid Kernel



Support Vector Machine Classification

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pplot as plt
from sklearn.svm import svc
from sklearn.model_selection import train_test_split

X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)

svc = SVC(kernel='linear')
svc.fit(X_train,y_train)
y_pred = svc.predict(x_test)

from sklearn.metrics import classification_report,confusion_matrix

print(classification_report,confusion_matrix)

from sklearn.model_selection import GridSearchCV *hyperparameter tuning

param_grid = {'C':[0.1,10,100,100],'gamma':[1,0.1,0.01,0.001,0.0001],'kernal':['rbf']} *parameters from svc()

grid = GridSearchCV(svc(),param_grid=param_grid,refit=True,verbose=True,cv=5)

grid(X_train,y_train)
y_pred = grid.predit(x_test)

grid.best_params_
print(classification_report(y_test,y_prod))
print(confusion_matrix(y_test,y_pred))
```

Support Vector Regression

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pplot as plt
from sklearn.svm import svc
from sklearn.model_selection import train_test_split
X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)

from sklearn.preprosessing import labelEncoder * to label encode, binary class
lel1 = labelEncoder()
lel2 = labelEncoder()
lel3 = labelEncoder()

import warnings
warnings.filterwarnings('ignore') *to avoid warning msg

X_train['sex'] = lel1.fit_transform(X_train['sex'])
X_train['smoker'] = lel2.fit_transform(X_train['smoker'])
X_train['time'] = lel3.fit_transform(X_train['time'])

x_test['sex'] = lel1.transform(x_test['sex'])
x_test['smoker'] = lel2.transform(x_test['smoker'])
x_test['time'] = lel3.transform(x_test['time'])

from sklearn.compose import ColumnTransformer
form sklearn.preprocessing import OneHotEncoder. * onehot encoding column transforme, for more than two category

ct = ColumnTransformer(transformers=[{'onnehot',OneHotEncoder(drop='first'),3},remaider='passthrough'])

X_train = ct.fit_transform(X_train)
x_test = ct.fit_transform(x_test)

from sklearn.svm import SCR  *model for support vector regression

svr = SCR()
svr.fit(x_train,y_train)

y_pred = svr.predict(x_test)

from skelearn.metrics import r2_score,mean_absoule_error
print(r2_score(y_test,y_pred))
print(mean_absoule_error(y_test,y_pred))

rom sklearn.model_selection import GridSearchCV  *hyperparameter Tunning using GriidSearchCV

param_grid = {'C':[0.1,10,100,100],'gamma':[1,0.1,0.01,0.001,0.0001],'kernal':['rbf']}
grid = GridSearchCV(svr(),param_grid=param_grid,refit=True,verbose=True,cv=5)

grid.fit(x_train,y_train)

grid_pred = grid.predit(x_test)

grid.best_params_
from skelearn.metrics import r2_score,mean_absoule_error
print(r2_score(y_test,grid_pred))
print(mean_absoule_error(y_test,grid_pred))
```

Naïve Baye's Algorithm (classification – binary. or multiclass classification)

P(A and B) = P(B and A)

P(A) * $P(A) * P(B/A) = P(B) * P\left(\frac{A}{B}\right)$

$$P(A/B) = \frac{P(A) * P(B/A)}{P(B)}$$

$$P\left(\frac{y}{(x_1,x_2,x_3)}\right) = \frac{P(y) * p(x_1,x_2,x_3)/y}{p(x_1,x_2,x_3)}$$ *$* \ x_1, x_2, x_3$ is independent and y is dependent variable*

$$\frac{p(y) * p\left(x_1/y\right) * p\left(x_2/y\right) * p\left(x_3/y\right)}{p(x_1,x_2,x_3)}$$

$$P\left(\frac{yes}{(x_1,x_2,x_3)}\right) = \frac{p(yes) * p\left(x_1/yes\right) * p\left(x_2/yes\right) * p\left(x_3/yes\right)}{p(x_1,x_2,x_3)}$$

$$P\left(\frac{no}{(x_1,x_2,x_3)}\right) = \frac{p(no) * p\left(x_1/no\right) * p\left(x_2/no\right) * p\left(x_3/no\right)}{p(x_1,x_2,x_3)}$$

Variants of Naïve Bayes

- Bernoulli Naïve Bayes
- Multinomial Naive Bayes
- Gaussian Naïve Bayes

Bernoulli Naïve Bayes
Whenever the independent features are following a Bernoulli distribution, then we need to use Bernoulli Naive Bayes algorithm (exp: the outcome (dependent features) should be 0 or 1, Yes or No, Pass or Fail .. only two outcomes)

Multinomial Naive Bayes
Whenever the independent features are TEXT, then use this algorithm (exp: spam classification, sentiment analysis)

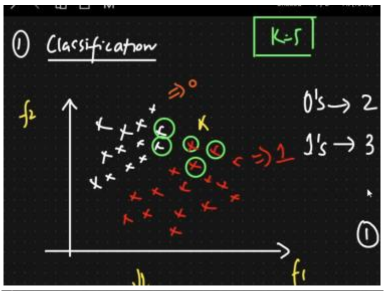
Gaussian Naïve Bayes
If the features are following Gaussian distribution, then we use Gaussian Naïve Bayes (features are continuous)

```
from sklean.model_selection import train_test_split
X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state = 42)

from sklean.naive_bayes import GaussianNG
```
**** we selected GaussianNG model because independent features (sepal height,widght) are continuous***

```
gng = GaussianNG()
gng.fit(X_train,y_train)

y_pred = gng.predict(x_test)

from sklearn.metrics import accuracy_score,classiffication_report,confusion_matrix

print(classiffication_report(accuracy_score(y_test,y_pred)y_test,y_pred),confustion_matrix(y_test,y_pred))
```

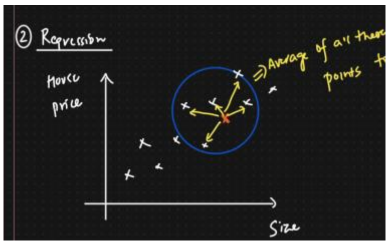
K Nearest Neighbor (KNN – classification (binary or multiclass), regression**)**

- Initiate the K value, k > 0 …. ∞, K = 1,2,3,4… **** K is hyperparameter***
- Find the K Nearest Neighbor for the test data
- From those K = 5 how many neighbors belongs to 0 and d1 category



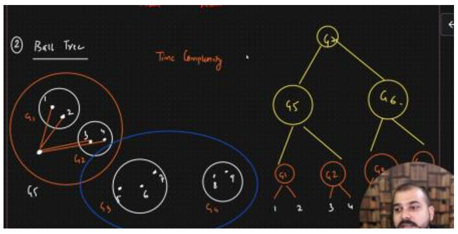
Points to follow when we introduce new points to classify in KNN classification

1. Select K value, which mean how many points to select as a neighbor of the newly introduced point.
2. Calculate the distance between the newly introduced point to the K neighbor points
3. Calculate how may points (K neighbor points) closed to newly introduced point
4. The newly introduced point will be assigned to the category, in which sum of number of closed points is high to the newly introduced point.



How to find the distance between points

1. Euclidian Distance $ED = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
2. Manhattan Distance $MD = |(x_2 - x_1)| + |(y_2 - y_1)|$
   - KNN will be impacted when the data set is biased (100$_{yes}$, 10$_{no}$)
   - KNN will be impacted when data set has outliers.
   - KNN with regression the newly introduced point will be calculated by the mean of neighbor points



KNN Variant

- KD Tree
- Ball Tree

```
import pandas as pd
import numpy as np
import matplotlib.pyplt as mp
from sklean.model_selection import train_test_split

X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state = 42)
```

```
from sklean.neighbors import KNeiighborsClassifier
gng = KNeiighborsClassifier(n_neibours = 5,algorith='quto')
gng.fit(X_train,y_train)

y_pred = gng.predict(x_test)

from sklearn.metrics import accuracy_score,classiffication_report,confusion_matrix

print(classiffication_report(accuracy_score(y_test,y_pred)y_test,y_pred),confustion_matrix(y_test,y_pred))
```

Decision Tree (classifier and regression)

- Purity – Pure or Impure(leaf node) split(no leaf node) (Entropy, Gini Impurity)
- Which feature should select for splitting (information gain)

Entropy

In the Binary classification (Decision Tree), it is important to identify the feature to begin splitting of nodes in decision tree, Entropy helps to measure the purity of the splits to reach the leaf node quickly from root node.

$Entropy\colon\ H(s) = -p_{(+)}\log_2(p_+) - p_{(-)}\log_2(p_-)$

where $p_{(+)}, p_{(-)}$ are Probability with % of +ve and % -ve class

Exp: node f_1 has $3_{yes}\ and\ 2_{no}$ then split with the Entropy – $\frac{3}{5}\log_2\frac{3}{5} - (\frac{2}{5}\log_2\frac{2}{5})$

Entropy value ranges between 0 to 1, lesser the value better the Entropy

Gini Impurity

Gini Impurity also helps to find purity of the split in binary classification like Entropy, but most of time Gini impurity is better than Entropy because of computational performance (No computational time for log)

$$GI = 1 - \sum_{i=1}^{n} p^2$$

gini impurity value ranges between 0 to 0.5, lesser the value better the gini impurity

When to use Entropy, Gini Impurity

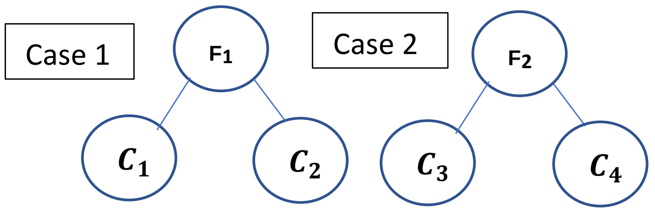
$Entropy\colon\ H(s) = -p_{(+)}\log_2(p_+) - p_{(-)}\log_2(p_-)$
$Gini\ Impurity = 1 - \sum_{i=1}^{n} p^2$
$for\ output\ 3\ catagory\ H(s) = -PC_1\ log_2 PC_1 - PC_2\ log_2 PC_2 - PC_3\ log_2 PC_3$

**** when the dataset is small then use Entropy***
**** when the dataset is not small then use Gini Impurity***

Information Gain

$Gain(s,a) = H(s) - \sum_{VE\ VAL}\frac{|S_v|}{|S|}H(S_v)$ ****H(s) root, $H(S_v)$ leaf category***

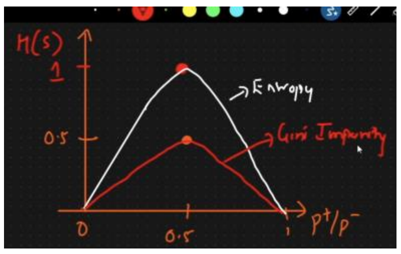
F₁ Root Node -> $H(s)$ = 9Y,5N
C_2 Leaf Node (sub set) -> $H(S_v)$ = 6Y,2N (impure split)
C_3 Leaf Node (sub set) -> $H(S_v)$ = 3Y,3N (50 %Impure split)
$|S|$ Total sample
$|S_v|$ Sample after the split



We get the following values after computing these values in

$H(s) = -p_{(+)}\log_2(p_+) - p_{(-)}\log_2(p_-)$

H(s) for $F_1 = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}\ log_2\frac{5}{14}\ *\boldsymbol{root\ note}$
$Entropy\ H(s)$ for F₁ = 0.94



H(S_v) for $C_1 = -\frac{6}{8}log_2\frac{6}{8} - \frac{2}{8}\ log_2\frac{2}{8}$
$Entropy\ H(S_v)$) for C_1 = 0.81

$Entropy\ H(S_v)$) for C_2= 1 ****because it has 50% impure split***

Information Gain (s, F₁) = $H(F_1) - \frac{8}{14}H(C_1)$ - $\frac{8}{14}H(C_2)$
Gain (s, F₁) = 0.94 – $\frac{8}{14}$ 0.81 - $\frac{8}{14}$ 1 => 0.049

For case1 we got information gain of 0.049, The information gain logic will calculate for case2 and so on until it gets the highest value. The root feature in split structure which gives the highest information gain value will consider for start node
e (Root Node) in binary classification.

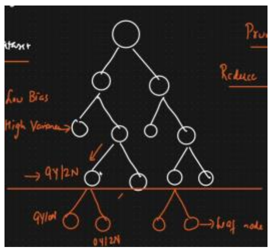
Decision Tree Split for Numerical Feature

Consider the following when use Decision Tree split for Numerical Feature

1. Sorting of Numerical Feature
2. Create Threshold value for Numerical Feature

Disadvantage of using Decision Tree split for Numerical Feature is, If the number of Numerical Feature is large, it takes longer time for computing Gini or Entropy impurity and Information Gain for each and every threshold value.

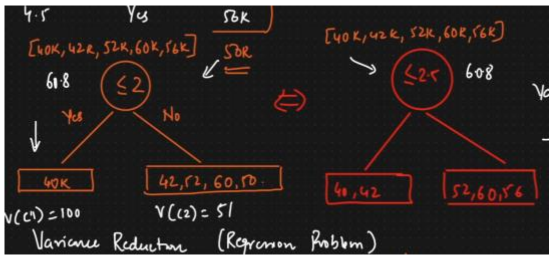
Post Pruning and Pre Pruning



In training when we completely split the data until we get leaf node (pure split) it leads into overfitting, this may impact while testing the data (low bias, high variance) . In order to avoid this, we use ‘post, pre pruning’.

In Post Pruning, we construct the decision tree, the prune it with respect to depth (apply for smaller dataset)
In Pre Pruning, hyperparameter (max features, max depth, split) tuning while constructing decision tree

Decision Tree Regression



In Decision Tree Regression we cannot use Entropy, Gini Impurity, Information Gain because the output (dependent feature is continuous)

Variance Reduction

$$\sigma^2 = \sum_{i=1}^{N} \frac{(x_i - \mu)^2}{N}$$

Variance Reduction = $Var(root) - \sum w_i \ Var(chid)$

$* \boldsymbol{w_i}$ ***ratio between the root and child***

```
import pandas pd
import numpy np
import matplotlib.pyplot plt
import sklearn.model_selection import train_test_split

X_train,x_test,y_train,y_test =train_test_split(X,y,test_size = 0.25,random_state=42)

from skllearn.tree import DecisionTreeClassifier

treeclassifier = DecisionTreeClassifier()
treeclassifier.fit(X_train,y_train)

from sklearn.import tree
plt.figure(sigsize = (15,10))
tree.plt_tree(treeclassifier,filled = True)

y_pred = treeclassifier.predict(x_test)

from sklean.metrics import confusion_matriix,classification_report
print(confusion_matriix(y_test,y_pred),classification_report(y_test,y_pred))
```

****preprunning and hyperparameter tuning***

```
param = {
        'criterion' : ['gini','entropy','log_loss'],
        'splitter':['best','random'],
        'max_depth':[1,2,3,4,5],
        'max_features':['auto','sqrt','log2']
}

treemodel = DecisionTreeClassifier()
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(tremodel,param_grid=param,cv=5,scoring='accuracy')

grid.fit(X_train,y_train)

grid.best_params_
grid.best_score_

y_pred = grid.predict(x_test)

from sklean.metrics import confusion_matriix,classification_report,accuracy_score
print(confusion_matriix(y_test,y_pred),classification_report(y_test,y_pred),accuracy_score(y_test,y_pred))
```

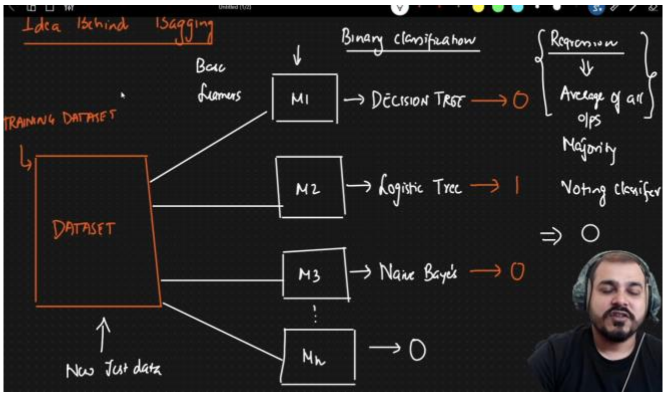
Ensemble Technique Bagging and Boosting

Ensemble means combining more than one models or technique. In classification problem Ensemble have two methods 1. Bagging 2. Boosting

Bagging (Bootstrap Agg)	**Boosting**
Random Forest	ADA Boost
	Gradient Boosting
	XgBoost

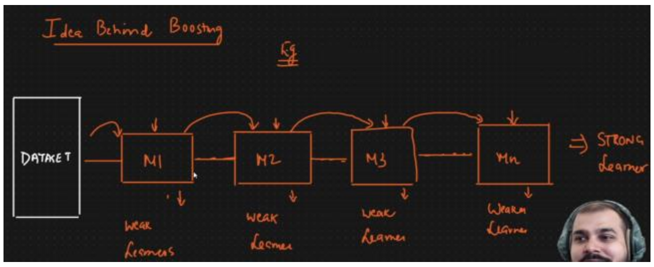
In the bellow figure sample data from data set will be distributed to different models with ‘Row Sampling with Replacement’ technique. Row sampling with Replacement means, It get sample data from data set and send (Boost) to model 1, when it send sample data to model 2, it sends send’s some of the data from the previous data (ie., to model1). It’s kind of overlapping at the edge of sampled data. In aggregation, It takes the highest value out of classification model, but in the case of regression (continues value) it takes the mean of the model’s output.

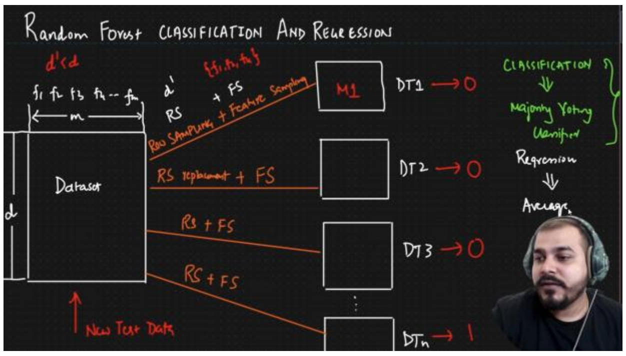
Bagging – trained parallelly (Random Forrest Alg



Boosting (serial trainer)

- ADA Booting
- Gradient Boosting
- XgBoost





Random Forest Classification

```
#Data Collection
#import libraries

import pandas pd
import numpy as np
import matplotlib.pyplot as plt
import seabornn as sns
import plotly.express as pxx
import warnings

warnings.filterwarnings('ignore')
%matplotlib inline

df = pd.read_csv('travel.csv')
df.head()

DATA CLEANING
Handling Missing Values
            1.Handling Missing Values
            2.Handling Duplicates
            3.Check data types
            4.Understand the dataset

            #check all the categories
df.isnull().sum()
df['Gender'].value_counts()
df['MaritalStatus'].value_counts()
df['TypeofContact'].value_counts()

df['Gender'] = df['Gender'].replace('Fe Male','Female')
df['MaritalStatus'] = df['MaritalStatus'].replace('Singlel','Unmarried')

#check missing values
#these are the features with nan value

features_with_na = [features for features in df.columns if df[feature].isnull().sum()>=1 ]
for  feature in features_with_na:
            print(feature,np.round(df[feature].isnull().mean()*100,5), '% missing value')

#statistics on numerical colums (null cols)
df[features_with_na].select_dtypes(exclude='object').describe()

Imputing Null Values
1. Impute Median value for Age column
2. Impute Mode for Type of Contract
3. Impute Median for Duration of Pitch
4. Impute Mode for NumberofFolllowup as it is Discrete feature
5. Impute Mode for PreferredPropertystar
6. Impute Median for NumberofTips
7. Impute Mode for NumberOfChildrenVisiting

#Age
df.Age.fillna(df.Age.median(),inplace=True)
```

```
#TypeofContract
df.TypeofContract.fillna(df.TypeofContact.mode()[0],inplace=True)
#DurationOfPitch
df.DurationOfPitch.fillna(df.DurationOfPitch.median(),inplace=True)
#NumberofFollowups
df.NumberOfFollowups.fillna(df.NumberOfFollowups.mode()[0],inplace=True)
#PreferredPropertyStar
df.PreferredPropertyStar.fillna(df.PreferredPropertyStar.mode()[0],inplace=True)
#NnumberofTrips
df.NumberofTrips.fillna(NumberofTrips.median(),inplace=True)
#NumberofChildrenVisiting
df.NumberofChildrenVisiting.fillna(df.NumberofChildrenVisiting.mode()[0],inplace=True)
#MonthlyIncome
df.MonthlyIncome.fillna(df.MonthlyIncome.median(),inplace=True)
df.drop('CustomerId',inplace=True,axis=1)
```

Feature Engineering
Feature Extraction

```
#create new column for feature
df['TotalVisiting'] = df[NumberOfPersonVisiting] + df['NumberofChildrenVisiting']
df.drop(columns=['NumberOfPersonVisiting','NumberofChildrenVisiting'],axis=1,inplace=True)

#get all the numeric features
num_features = [feature for feature in df.columns if df[feature].dtype !='O']
print('Num of Numerical Features', len(num_reatures))

# get all the categorical features
cat_features = [feature for feature in df.columns if df[feature].dtype ='O']
print('Num of Categorical Features', len(cat_features))

#get discrete features
discrete_features = [feature for feature in num_features if len(df[feature].unique() <= 25)]
print('Num of Categorical Features', len(discrete_features))

# get continuous features
continuous_features = [feature for feature in  num_features if feature not in                    discrete_features )]
print('Num of Categorical Features', len(continuous_features))

#Train Test spliit and Model Training
from sklearn.model_selection import train_test_split
X = df.drop['ProdTaken',axis =1]
y= df['ProdTaken']

y.value.count()
X.head()

#separate dataset into train and test
X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)

X_train.shape,x_test.shape

#create column transformer with 3 types of transformers
cat_features = X.select_dtypes(include ='objects').columns
num_features = X.select_dtypes(include ='objects').columns

from sklean.prepocessing import OneHotEncoder,StadardScalar
from sklean.compose import ColumnTranformer

numeric_transformer = StandardScalar()
oh_transformer = OneHotEncoder(drop='fist')

preprocessor = ColumnTransformer(
                                                    [
                                                            ('OneHotEncoder',oh_transformer,cat_features),
                                                            ('StandardScalaar',numeric_transformer,num_features)
                                                    ]
                                                    )

X_train = preprocessor.fit_transform(X_train)
x_test= preprocessor.transform(x_test)

#Machine Learning Training

from sklearn.ensomble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report,ConfusionMatrisDisplay,precision_score,
recall_score,f1_score,roc_auc_score,roc_curve


models = {
                    'Random Forest':RandomForestClassifier()
                    'Decision Tree':DecisionTreeClassifier()
                    'Logistic Regression':LogisticRegression()
            }

for i in range(len(list(models))):
        model = list(models.values())[i]
        model.fit(X_train,y_train) #train model

        #make prediction
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(x_test)

        #training set performance
        model_train_accuracy = accuracy_score(y_train,y_train_pred)
        model_train_f1 = f1_score(y_train,y_train_pred,average='weighted')
        model_train_precision = precision_score(y_train,y_train_pred)
        model_train_recall = recall_score(y_train,y_train_pred)
        model_train_rocauc_score = roc_auc_score(y_train,y_train_pred)
```

```
            #test set performance
            model_test_accuracy = accuracy_score(y_test,y_test_pred)
            model_test_f1 = f1_score(y_test,y_test_pred,average='weighted')
            model_test_precision = precision_score(y_test,y_test_pred)
            model_test_recall = recall_score(y_test,y_test_pred)
            model_test_rocauc_score = roc_auc_score(y_test,y_test_pred)

            print(list(models.keys()[i]))

            print('Model performance for Training set')
            print('- Accuracy:{:.4f}'.format(model_train_accuracy))
            print('- F1 Score:{:.4f}'.format(model_train_f1))
            print('- Precision:{:.4f}'.format(model_train_precision))
            print('- Recall:{:.4f}'.format(model_train_recall))
            print('- Accuracy:{:.4f}'.format(model_train_accuracy))

            print(f'------------------------')
            print('Model performance for Test set')
            print('- Accuracy:{:.4f}'.format(model_test_accuracy))
            print('- F1 Score:{:.4f}'.format(model_test_f1))
            print('- Precision:{:.4f}'.format(model_test_precision))
            print('- Recall:{:.4f}'.format(model_test_recall))
            print('- Accuracy:{:.4f}'.format(model_test_accuracy))

#Hyperparameter Training
rf_param = {
                        'max_depth':[5,8,15,None,10],
                        'max_features':[5,7,'auto',8],
                        'min_sample_split':[2,8,15,20],
                        'n_estimators':[100,200,500,1000]
                        }

# models list for hyperparameters tuning

randomcv_model = [
                                    ('RF',RandomForestClassifier(),rf_params),

                                   ]

from sklearn.model_selection import RandomizedSearchCV

model_param = {}

for name,model,params in randomcv_models:
            random = RandomizedSearchCV(estimator=model,param__distribution=params,nn_iter=100,cv=3,verbose=2,n_jobs=-1)
            random.fit(X_train,y_train)
            model_param[name] = random.best_params_

for model_name in model_param:
            print(f'------Best Param for {model_name}------')
            print(model_param[model_name])


#model trainig after parameter optimization using randamizeCV

modesl = (
                        'Random Fores':RandomForestClassifier(n_estimators=1000,min_samples_split=2,max_featuers=7,max_dept=Nore)
                  )
for i in range(len(list(models))):
            model = list(models.values())[i]
            model.fit(X_train,y_train) #train model

            #make prediction
            y_train_pred = model.predict(X_train)
            y_test_pred = model.predict(x_test)

            #training set performance
            model_train_accuracy = accuracy_score(y_train,y_train_pred)
            model_train_f1 = f1_score(y_train,y_train_pred,average='weighted')
            model_train_precision = precision_score(y_train,y_train_pred)
            model_train_recall = recall_score(y_train,y_train_pred)
            model_train_rocauc_score = roc_auc_score(y_train,y_train_pred)

            #test set performance
            model_test_accuracy = accuracy_score(y_test,y_test_pred)
            model_test_f1 = f1_score(y_test,y_test_pred,average='weighted')
            model_test_precision = precision_score(y_test,y_test_pred)
            model_test_recall = recall_score(y_test,y_test_pred)
            model_test_rocauc_score = roc_auc_score(y_test,y_test_pred)

            print(list(models.keys()[i]))

            print('Model performance for Training set')
            print('- Accuracy:{:.4f}'.format(model_train_accuracy))
            print('- F1 Score:{:.4f}'.format(model_train_f1))
            print('- Precision:{:.4f}'.format(model_train_precision))
            print('- Recall:{:.4f}'.format(model_train_recall))
            print('- Accuracy:{:.4f}'.format(model_train_accuracy))

            print(f'------------------------')
            print('Model performance for Test set')
            print('- Accuracy:{:.4f}'.format(model_test_accuracy))
            print('- F1 Score:{:.4f}'.format(model_test_f1))
            print('- Precision:{:.4f}'.format(model_test_precision))
            print('- Recall:{:.4f}'.format(model_test_recall))
            print('- Accuracy:{:.4f}'.format(model_test_accuracy))

# plot ROC AUC curve
from sklearn.metrics import roc_auc_score,roc_curve
plot.figure()
```

```
#add the models to the list that you want to view on the ROC plot

auc_models = [
                                        {
                                                'label':'Random Forest Classifier',
                                                'model': RandomForestClassifier(n_estimators=200,min_samples_split=2,max_features=7,max_depth=15),
                                                'auc':0.8319
                                        }
                                        ]

#create loop through all model
for algo in auc_models:
        model=algo['model'] #select the model
        model.fit(X_train,y_train) #train the model
        #comput false positive rate, and True positive rate
        fpr,tpr,thresholds = roc_curve(y_test,model.predict_praba(x_test)[:,])
        #calculate area under the curve to display on the plot
        plt.plot(fpr,tpr,label='%s' ROC (area=%0.2f)'%(algo['label']))
        #custom settings ffor the plot
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('1-Specificity(False Positive Rate')
plt.ylable('Sensitiivity(True Positivie Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.savefig('auc.png')
plt.show()
```

Random Forest Regression

```
#Data Collection
#import libraries

import pandas pd
import numpy as np
import matplotlib.pyplot as plt
import seabornn as sns
import plotly.express as pxx
import warnings

warnings.filterwarnings('ignore')
%matplotlib inline

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

DATA CLEANING
Handling Missing Values
1.Handling Missing Values
2.Handling Duplicates
3.Check data types
4.Understand the dataset

```
#check all the categories
df.isnull().sum()

# remove unnecessary columns
df.drop('car_name',axis = 1,inplace=True)
df.drop('brand',axis=1,inplace=True)

df.head()
df['model'].unique()
```

Imputing Null Values
1. Impute Median value for Age column
2. Impute Mode for Type of Contract
3. Impute Median for Duration of Pitch
4. Impute Mode for NumberofFolllowup as it is Discrete feature
5. Impute Mode for PreferredPropertystar
6. Impute Median for NumberofTips
7. Impute Mode for NumberOfChildrenVisiting

Feature Engineering
Feature Extraction

```
#create new column for feature
df['TotalVisiting'] = df[NumberOfPersonVisiting] + df['NumberofChildrenVisiting']
df.drop(columns=['NumberOfPersonVisiting','NumberofChildrenVisiting'],axis=1,inplace=True)

#get all the numeric features
num_features = [feature for feature in df.columns if df[feature].dtype !='O']
print('Num of Numerical Features', len(num_reatures))

# get all the categorical features
cat_features = [feature for feature in df.columns if df[feature].dtype ='O']
print('Num of Categorical Features', len(cat_features))

#get discrete features
discrete_features = [feature for feature in num_features if len(df[feature].unique() <= 25)]
```

```
print('Num of Categorical Features', len(discrete_features))

# get continuous features
continuous_features = [feature for feature in  num_features if feature not in                    discrete_features )]
print('Num of Categorical Features', len(continuous_features))

#Train Test spliit and Model Training
from sklearn.model_selection import train_test_split
X = df.drop['selling_price',axis =1]
y= df['selling_price']

y.value.count()
X.head()

#separate dataset into train and test
X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)

X_train.shape,x_test.shape

from sklearn.preprocessing import LaabelEncoder
le = LaabelEncoder()
X['model'] =le.fit_tranform(S['model'])

#create column transformer with 3 types of transformers
num_features = X.select_dtypes(include ='objects').columns
onehot_columns = ['seller_type','fuel_type','transmission_type']
label_encoder_columnns = ['model']


from sklearn.preprocessing import OneHotEncoder,StadardScalar
from sklearn.compose import ColumnTranformer

numeric_transformer = StandardScalar()
oh_transformer = OneHotEncoder(drop='fist')

preprocessor = ColumnTransformer(
                                                                                        [
                                                                                                  ('OneHotEncoder',oh_transformer,cat_features),
                                                                                                  ('StandardScalaar',numeric_transformer,num_features)
                                                                                        ]
                                                                                        ),remainder='passthrough'

X_train = preprocessor.fit_transform(X_train)
x_test= preprocessor.transform(x_test)

#Machine Learning Training

from sklearn.ensomble import RandomForestRegressor
from sklearn.neighbors import KNNeighborsRegressor
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score,mean_absolute_error,mean_square_error

#create a function to evaluate model
dev evaluate_model(true,predicted):
         mae = mean_absolute_error(true,predicted)
         mse = mean_square_error(true,predicted)
         rmse = np.sqrt(mean_square_error(true,predicted))
         r2_square = r2_score(true,predicted)
         return mae,rmse,r2_square

models = {
                           'Linear Regression' : RandomForestRegressor()
                           'Lasso' : Lasso()
                           'Ridge' : Ridge()
                           'K-Neighbors Regressor': KNNeighborsRegressor()
                           'Decisiton Tree': DecisionTreeRegressor()
                           'Random Forest Regression': RandomForestRegressor()
                  }

for i in range(len(list(models))):
         model = list(models.values())[i]
         model.fit(X_train,y_train) #train model

         #make prediction
         y_train_pred = model.predict(X_train)
         y_test_pred = model.predict(x_test)

         # evaluate train and test dataset
         model_train_mae,model_train_rmse,model_train_r2 = evaluate_model(y_train_pred,X_train)
         model_test_mae,model_test_rmse,model_test_r2 = evaluate_model(y_test_pred,x_test)


         print(list(models.keys())[i])

         print('Model performance for training set')
```

```
                print('Root Mean Squuared Error: {:,4f}'.format(model_train_r2)
                print('Mean Absolute Error: {:.4f}'.format(model_train_mae)
                print('R2 Score: {:.4f}'.format(model_train_rmse))

                print(f'---------------')

                print('Model performance for test set')
                print('Root Mean Squuared Error: {:,4f}'.format(model_test_r2)
                print('Mean Absolute Error: {:.4f}'.format(model_test_mae)
                print('R2 Score: {:.4f}'.format(model_test_rmse))

# initialize few hyperparameter tuning

knn_param = {'n_neibors': [2,3,10,20,40,50]
                                            }
rf_param = { 'max_features':[5,7,'auto',8],
                  'max_depth':[5,8,15,None,10],
                 'min_samples_split':[2,8,15,20],
                ''n_estimators':[100,200,500,1000]
                  }
# models list for Hyperparameter tuning

randomcv_model = [
                        ('KNN',KNeighborsRegressor(),knn_param),
                        ('RF',RandomForestRegressor(),rf_param)
                 ]

#Hyperparameter Tuning
from sklearn.model_selection import RandomizedSearchCV

model_param{}
for name,model,params in randomcv_model:
                random = RandomizedSearchCV(estimator = model,param_distributiions = params,n_iter=100,cv=3,verbose=2,n_jobs=-1)
                random.fit(X_train,y_train)
                model_param[name] = random.best_params_

for model_name in model_param:
                print(f'----------- Best Params for {moedl_name}---')
                print(model_param[model_name])

# Retraining the model with best parameters

models = {
                'Random Forest Regressor':RandomForestRegressor(n_estimators =100,min_samples_split = 2,max_feature='auto',max_depth= None,h_jobs=1),
                'K-Neighbors Regressor': KNeighborsRegressor(n_neighbors=10,n_jobs=-1)
                }

for i in range(len(list(models))):
                model = list(models.values())[i]
                model.fit(X_train,y_train) #train model

                #make prediction
                y_train_pred = model.predict(X_train)
                y_train_pred = model.predict(x_test)

                #evaluate train and test dataset
                model_train_mae,model_train_rmse,model_train_r2 = evaluate_model(y_train,X_train)
                model_test_mae,model_test_rmse,model_test_r2 = evaluate_model(y_test,X_test)

                print(list(models.keys())[i])

                print('Model performance for training set')
                print('Root Mean Squuared Error: {:,4f}'.format(model_train_r2)
                print('Mean Absolute Error: {:.4f}'.format(model_train_mae)
                print('R2 Score: {:.4f}'.format(model_train_rmse))

                print(f'---------------')

                print('Model performance for test set')
                print('Root Mean Squuared Error: {:,4f}'.format(model_test_r2)
                print('Mean Absolute Error: {:.4f}'.format(model_test_mae)
print('R2 Score: {:.4f}'.format(model_test_rmse))
```

What is Boosting:

- Boosting works by learning from previous mistakes (errors in model predictions) to come up with better future predictions
- Boosting is an ensemble machine learning technique that works by training weak models in a sequential fashion
- Boosting algorithms work by building a model from the training data, then the second model is built based on the mistakes (residuals) of the first model. The algorithm repeats until the maximum number of models have been created or until the model provides good predictions.

What is Ensemble Learning:

- XGBoost is an example of ensemble learning

- Ensemble techniques such as bagging and boosting can offer an extremely powerful algorithm by combining a group of relatively weak/average ones
- For example, you can combine several decision trees to create a powerful random forest algorithm
- Boosting can reduce variance and overfitting and increase the model robustness

Advantages:
- No need to perform any feature scaling
- Can work well with missing data
- Robust to outliers in the data
- Can work well for both regression and classification
- Computationally efficient and produce fast predictions

Disadvantages:
- Poor extrapolation characteristics
- Need extensive tuning
- Slow training

AdaBoost (classification, regression)
AdaBoost follows the steps
1. Create Sample weight $w = \frac{1}{n}$ where n is total rows in the dataset.
2. Create base learns to find total error, In AdaBoost all base leaners are decision tree. Create decision tree with one depth (one parent node with two leaf nodes (stumps)) and get the entropy or Gini coefficient value for purity of splits.
3. Performance of Stump $\frac{1}{2}\log_c(\frac{1-TE}{TE})$ $TE\ Total\ Error * \boldsymbol{ie., \alpha_1}$
4. Update Weight for incorrect points, ie., increase the weight of incorrect points $New\ sampple\ weight = Weight * e^{performance\ of\ stump}$
5. Update Weight for correct points, ie., decrease the weight of correct points $New\ sampple\ weight = Weight * e^{-\ performance\ of\ stump}$

$$f = \alpha_1(m_1) + \alpha_2(m_2) + \alpha_3(m_3) + \ldots . \alpha_n(m_n)$$

***$\boldsymbol{\alpha_1}$ weights, m decision tree stumps (model result)**

```
from sklearn.ensemble import AdaBoostClassifier
```

Gradient Boosting (classification, regression)

1. Create Base Model
2. Find residual value
3. Construct Decision Tree

Gradient Boosting => Base Model -> Decision Tree 01 -> Decision Tree 02 .. -> Decision Tree n

$$F(x) = h_0(x) + \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_n h_n(x)$$
$$were\ \alpha\ is\ learning\ rate, which\ ranges\ from\ 0\ to\ 1$$
$$F(x) = \sum_{i=1}^{n} \alpha_n h_n(x)$$

Required Task for Gradient Boosting Algorithm
1. Provide Input and Output value (Input values – Exp, Degree, Output values – Salary)
2. Provide loos functions (for Regression – RME, MSE, for Classification – Hinge)
3. Provide how many Binary Tree needed
4. Sudo Algorithm
   - Initialize Model with constant value, for the base model.
   - First order derivative for $y^n\ is\ \frac{\partial y}{\partial x} = n\, y^{n-1}$, equation of lose function $\sum_{i=1}^{n} \frac{1}{2}(y - y\hat{})^2$
   - $F_0(x) = argMinargMin_\gamma \sum_{i=1}^{n} Lossfn(y, \gamma)$
   - Iterate M =1 to M
   - Compute Pseudo residuals $r_{im} = -\left[\frac{dl\,(y, F(x_i)}{dF(x_i)}\right]$
   - Fit a base learner $h_m(x)$ where input values are $\{x_i, r_{im}\}$ were r_i is difference between (salary) output value and constant value.

Gradient Boosting Classification

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DeecisionTreeCllassifier
from sklearn.lener_model import Logistic_regressioin
from sklearn.metrics import accuray_score,claassificaatiion_report,ConfusionMetrix,preciision_score,recall_score,f1_score,roc_auc_score

models = {
        'Logistic Regression':Logistic_regressioin(),
        'Decision Tree' : DeecisionTreeCllassifier(),
        'Random Forest' : RandomForestClassifier(),
        'Gradieent Boost': GradientBoostingClassifier(),
        'Adaboost':AdaBoostClassifier()
}

for i in range(len(list(models))):
        model == list(models.values())[i]
        model.fit(X_train,y_train)

        #make prediction
        y_train_pred = model.prediction(x_train)
        y_test_pred = model.prediction(x_test)

        #training set performance
        model_train_accuracy = accuray_score(y_train,y_train_pred)
        model_train_f1 = f1_score(y_train,y_train_pred)
        model_train_precision = preciision_score(y_train,y_train_pred)
        model_train_recall = recall_score(y_train,y_train_pred)
        modell_train_rocauc_score = roc_auc_score(y_train,y_train_pred)

        #test set performance
        model_test_accuracy = accuray_score(y_test,y_test_pred)
        model_test_f1 = f1_score(y_test,y_test_pred)
        model_test_precision = preciision_score(y_test,y_test_pred)
        model_test_recall = recall_score(y_test,y_test_pred)
        modell_test_rocauc_score = roc_auc_score(y_test,y_test_pred)

        print(list(models.keys()))[i]
        print('Model performannce for Training set')
        print(f'- Accuracy:{model_train_accuracy}')
        print(f'- F1 Score:{model_train_f1}')
        print(f'- Precision:{model_train_precision}')
        print(f'- Recall:{model_train_recall}')
        print(f'- Roc Auc Score:{modell_train_rocauc_score}')

        print('Model performannce for Test set')
        print(f'- Accuracy:{model_test_accuracy}')
        print(f'- F1 Score:{model_test_f1}')
        print(f'- Precision:{model_test_precision}')
        print(f'- Recall:{model_test_recall}')
        print(f'- Roc Auc Score:{modell_test_rocauc_score}')

        # Hyperparameter Training
        rf_params = {            'max_dept'=[5,8,15,None,10],
                                 'max_features': [5,7,'auto',8],
                                 'min_sample_split':[2,8,15,20],
                                 'n_estimators':[100,200,500,1000]}

        gradient_params = {      'loss':['log_loss','deviance','exponential'],
                                 'criterion':['friedman_mse','squared_error','mse'],
                                 'n_estimators':[100,200,500],
                                 'max_depth':[5,8,15,None,10]}

        #Models list for Hyperparameter turning
        randomcv_models = [('RF',RandomForestCllassifer(),rf_params),
                ('GradientBoost',GradientBoostingClassifier(),gradient_params)                                        ]

        from sklearn.model_seletion import RandomizedSearchCV

        model_param ={}
        for name,model,params in randomcv_models:
                random = RandomizedSearchCV(estimator=model,param_distributions = params,n_iter=100,cv=3,verbose=2,n_jobs=-1)

                random.fit(X_train,y_train)
                model_param[name] = random.best_params_

for model_name in model_param
        print(f'--------- Best param for {model_name} ---)
        print(model_param[model_name])

models = {
                'Random Forest':RandomForestClassifier(n_estimators=1000,min_samples_spl,max_features=7,max_depth=None),
                'GradientBoostclassifiier':GradientBoostingClassifier(n_estimators=500,min_samples_split=20,max_depth=15,loss='exponential',criterion='mse')
}

for i in range(len((list(models)))):
        model = list(models.vallue())[i]
        model.fit(X_train,y_train) #train model

        #make predictions
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(x_test)

        #traning set performance
        #training set performance
        model_train_accuracy = accuray_score(y_train,y_train_pred)
        model_train_f1 = f1_score(y_train,y_train_pred)
        model_train_precision = preciision_score(y_train,y_train_pred)
        model_train_recall = recall_score(y_train,y_train_pred)
        modell_train_rocauc_score = roc_auc_score(y_train,y_train_pred)

        #test set performance
        model_test_accuracy = accuray_score(y_test,y_test_pred)
        model_test_f1 = f1_score(y_test,y_test_pred)
        model_test_precision = preciision_score(y_test,y_test_pred)
        model_test_recall = recall_score(y_test,y_test_pred)
        modell_test_rocauc_score = roc_auc_score(y_test,y_test_pred)

        print(list(models.keys()))[i]
        print('Model performannce for Training set')
        print(f'- Accuracy:{model_train_accuracy}')
        print(f'- F1 Score:{model_train_f1}')
        print(f'- Precision:{model_train_precision}')
        print(f'- Recall:{model_train_recall}')
        print(f'- Roc Auc Score:{modell_train_rocauc_score}')

        print('Model performannce for Test set')
        print(f'- Accuracy:{model_test_accuracy}')
        print(f'- F1 Score:{model_test_f1}')
        print(f'- Precision:{model_test_precision}')
        print(f'- Recall:{model_test_recall}')
        print(f'- Roc Auc Score:{modell_test_rocauc_score}')

# plot ROC AUC curve

from sklearn.metrics import roc_auc_score,roc_cuurve
plt.figure()

# Add the models to the llist that you want to view on the ROc plot

auc_models = [
     {
      'lebel':'Gradient Boost Classifier',
      'model':GradientBoostingClassifier(n_estimators=500,min_samples_split=20,max_depth=15,loss='exponential',criterion='mse')
     }]

  # create loop throuugh all model
 for algo in auc_models:
        model == algo['model'] # select the model
        model.fit(X_train,y_trainn) #train the model
        #compute fals posiitiive rate and true positive rate
        fpr,tpr,thereshoulds = roc_cuurve(y_test,model.predict_proba(x_test)[:,1])
        #calculate Area under the curve to display on the plot
        plt.plot(fpr,tpr,label = '%s ROC (area = %0.2f' % (algo['label'],algo['auc'])
        #custom settiing for the plot
        plt.plot([0,1],[0,1],'r--')
        plt.xlim(0.0,1.0)
        plt.ylim(0.0,1.05)
        plt.xlabel('1-specificity(False Positive Rae')
        plt.ylabel('Sensitivitity(True Positive Rate')
        plt.title('Receiver Operating Characteristic')
        plt.legend(loc ='lower right')
        plt.savefig('auc.png')
        plt.show()
```

Gradient Boosting Regression

Xgboost Algorithm (classification, regression)
steps:

- Construct a base model
- Construct a Decision Tree with the root
- Calculate Similarity Weight for classification $\frac{\sum(Residueal)^2}{\sum \Pr(1-Pr)+\lambda}$
- Calculate Similarity Weight for regression $\frac{\sum(Residual)^2}{No.of\ Residual+\lambda}$
- Calculate Gain

Xgboost Algorithm Classification

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DeecisionTreeCllassifier
from xgboost import XGBClassifier
from sklearn.lener_model import Logistic_regressioin
from sklearn.metrics import accuray_score,claassificaatiion_report,ConfusionMetrix,preciision_score,recall_score,f1_score,roc_auc_score

models = {
        'Logistic Regression':Logistic_regressioin(),
        'Decision Tree' : DeecisionTreeCllassifier(),
        'Random Forest' : RandomForestClassifier(),
        'Gradieent Boost': GradientBoostingClassifier(),
        'Adaboost':AdaBoostClassifier()
        'Xgboost': XGBClassifier()
}

for i in range(len(list(models))):
        model == list(models.values())[i]
        model.fit(X_train,y_train)

        #make prediction
        y_train_pred = model.prediction(x_train)
        y_test_pred = model.prediction(x_test)

        #training set performance
        model_train_accuracy = accuray_score(y_train,y_train_pred)
        model_train_f1 = f1_score(y_train,y_train_pred)
        model_train_precision = preciision_score(y_train,y_train_pred)
        model_train_recall = recall_score(y_train,y_train_pred)
        modell_train_rocauc_score = roc_auc_score(y_train,y_train_pred)

        #test set performance
        model_test_accuracy = accuray_score(y_test,y_test_pred)
        model_test_f1 = f1_score(y_test,y_test_pred)
        model_test_precision = preciision_score(y_test,y_test_pred)
        model_test_recall = recall_score(y_test,y_test_pred)
        modell_test_rocauc_score = roc_auc_score(y_test,y_test_pred)

        print(list(models.keys()))[i]

        print('Model performannce for Training set')
        print(f'- Accuracy:{model_train_accuracy}')
        print(f'- F1 Score:{model_train_f1}')
        print(f'- Precision:{model_train_precision}')
        print(f'- Recall:{model_train_recall}')
        print(f'- Roc Auc Score:{modell_train_rocauc_score}')

        print('Model performannce for Test set')
        print(f'- Accuracy:{model_test_accuracy}')
        print(f'- F1 Score:{model_test_f1}')
        print(f'- Precision:{model_test_precision}')
        print(f'- Recall:{model_test_recall}')
        print(f'- Roc Auc Score:{modell_test_rocauc_score}')

        # Hyperparameter Tunning
        rf_params = {'max_dept'=[5,8,15,None,10],
                                'max_features': [5,7,'auto',8],
                                'min_sample_split':[2,8,15,20],
                                'n_estimators':[100,200,500,1000]}
        xgboost_params = {'learning_rate':[0.1,0.01],
                                        'max_depth': [5,8,12,20,30],
                                        'n_estimators': [100,200,300],
                                        'colsamplle_bytree':[0.5,0.8,1.0,0.4]  }
        #Models list for Hyperparameter turning
        randomcv_models = [('RF',RandomForestCllassifer(),rf_params),
                ('XgBoost',XGBClassifier(),xgboost_params)
                                                ]

        from sklearn.model_seletion import RandomizedSearchCV

        model_param ={}
        for name,model,params in randomcv_models:
                random = RandomizedSearchCV(estimator=model,param_distributions = params,n_iter=100,cv=3,verbose=2,n_jobs=-1)

                random.fit(X_train,y_train)
                model_param[name] = random.best_params_

for model_name in model_param:
        print(f'--------- Best param for {model_name} ---)
        print(model_param[model_name])

models = {
                'Random Forest':RandomForestClassifier(n_estimators=1000,min_samples_spl,max_features=7,max_depth=None),
                'xgboost':XGBClassifier(n_estimators=200,learning_rate=0.1,colsample_bytree=1)
}

for i in range(len((list(models)))):
        model = list(models.vallue())[i]
        model.fit(X_train,y_train) #train model

        #make predictions
        y_train_pred = model.predict(X_train)
        y_test_pred = model.predict(x_test)
```

```
                    #traning set performance
                    #training set performance
                    model_train_accuracy = accuray_score(y_train,y_train_pred)
                    model_train_f1 = f1_score(y_train,y_train_pred)
                    model_train_precision = preciision_score(y_train,y_train_pred)
                    model_train_recall = recall_score(y_train,y_train_pred)
                    modell_train_rocauc_score = roc_auc_score(y_train,y_train_pred)

                    #test set performance
                    model_test_accuracy = accuray_score(y_test,y_test_pred)
                    model_test_f1 = f1_score(y_test,y_test_pred)
                    model_test_precision = preciision_score(y_test,y_test_pred)
                    model_test_recall = recall_score(y_test,y_test_pred)
                    modell_test_rocauc_score = roc_auc_score(y_test,y_test_pred)

                    print(list(models.keys()))[i]
                    print('Model performannce for Training set')
                    print(f'- Accuracy:{model_train_accuracy}')
                    print(f'- F1 Score:{model_train_f1}')
                    print(f'- Precision:{model_train_precision}')
                    print(f'- Recall:{model_train_recall}')
                    print(f'- Roc Auc Score:{modell_train_rocauc_score}')

                    print('Model performannce for Test set')
                    print(f'- Accuracy:{model_test_accuracy}')
                    print(f'- F1 Score:{model_test_f1}')
                    print(f'- Precision:{model_test_precision}')
                    print(f'- Recall:{model_test_recall}')
                    print(f'- Roc Auc Score:{modell_test_rocauc_score}')

# plot ROC AUC curve

from sklearn.metrics import roc_auc_score,roc_cuurve
plt.figure()

# Add the models to the llist that you want to view on the ROc plot

auc_models = [
            {
             'lebel':'xgboost',
             'model':XGBClassifier(n_estimators=200,learning_rate=0.1,colsample_bytree=1)
            }]

  # create loop throuugh all model
 for algo in auc_models:
                    model == algo['model'] # select the model
                    model.fit(X_train,y_trainn) #train the model
                    #compute fals posiitiive rate and true positive rate
                    fpr,tpr,thereshoulds = roc_cuurve(y_test,model.predict_proba(x_test)[:,1])
                    #calculate Area under the curve to display on the plot
                    plt.plot(fpr,tpr,label = '%s ROC (area = %0.2f' % (algo['label'],algo['auc'])
                    #custom settiing for the plot
                    plt.plot([0,1],[0,1],'r--')
                    plt.xlim(0.0,1.0)
                    plt.ylim(0.0,1.05)
                    plt.xlabel('1-specificity(False Positive Rae')
                    plt.ylabel('Sensitivitity(True Positive Rate')
                    plt.title('Receiver Operating Characteristic')
                    plt.legend(loc ='lower right')
                    plt.savefig('auc.png')
                    plt.show()
```

Xgboost Algorithm Regression

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GraddianBoostingRegressor
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.heighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error

#create a function to evaluate model
def evaluate_model(true,predicted):
                    mae = mean_absolute_error(true,predicted)
                    mse = mean_squared_error(true,predicted)
                    rmse = np.sqrt(mean_squared_error(true,predicted))
                    r2_square = r2_score(true,predictedd)

# Beginning Model Train
model = {
                    'Linear Regerssion':LinearRegression(),
                    'Lasso' : Lasso(),
                    'Ridge' : Ridge(),
            'K-Neighbours Regressor': KNeighborsRegressor(),
            'Decision Tree': DecisionTreeRegressor(),
            'Randomm Forest Regressor': RandomForestRegressor(),
            'Agaboost Regressor': AdaBoostRegressor(),
            'Graident BoostRegressor':GraddianBoostingRegressor(),
            'Xgboost Regressor':XGBRegressor()
            }

for i in range(len(list(models))):
            model = list(models.value())[i]
            model.fit(X_train,y_train) #train model

            #make prediction
            y_train_pred = model.predict(X_train)
            y_test_pred = model.predict(x_test)

            #evaluate train and test dataset
            model_train_mae,model_train_rmse,model_train_r2 = evaluate_model(y_train,X_train)
            model_test_mae,model_test_rmse,model_test_r2 = evaluate_model(y_test,x_test)

            print(list(models.keys())[i])

            print('Model performance for training set')
            print('Root Mean Squuared Error: {:,4f}'.format(model_train_r2)
            print('Mean Absolute Error: {:.4f}'.format(model_train_mae)
            print('R2 Score: {:.4f}'.format(model_train_rmse))

            print(f'---------------')

            print('Model performance for test set')
            print('Root Mean Squuared Error: {:,4f}'.format(model_test_r2)
            print('Mean Absolute Error: {:.4f}'.format(model_test_mae)
            print('R2 Score: {:.4f}'.format(model_test_rmse))

# initialize few hyperparameter tuning

rf_param = {'max_depth': [5,8,15,None,10],
                        'max_feature':[5,7,'auto',8],
                        'min_samples_split':[2,8,15,20],
                        'n_estimators':[100,200,500,1000]
                        }
xgboost_param = { 'learning_rate':[0.1,0.01],
                  'max_depth':[5,8,12,20,30],
                 'n_estimators':[100,200,300],
                ''colsamplle_bytree':[0.5,0.8,1,0.3,0.4]
                  }
# models list for Hyperparameter tuning

randomcv_model = [
                    ('RF',RandomForestRegressor(),rf_param),
                    ('xgboost',XGBRegressor(),xgboost_param)
                 ]

#Hyperparameter Tuning
from sklearn.model_selection import RandomizedSearchCV

model_param{}
for name,model,params in randomcv_model:
            random = RandomizedSearchCV(estimator = model,param_distributiions = params,n_iter=100,cv=3,verbose=2,n_jobs=-1)
            random.fit(X_train,y_train)
            model_param[name] = random.best_params_

for model_name in model_param:
            print(f'----------- Best Params for {moedl_name}---')
            print(model_param[model_name])

# Retraining the model with best parameters

models = {
            'Random Forest Regressor':RandomForestRegressor(n_estimators =100,min_samples_split = 2,max_feature=5,max_depth= None,h_jobs=1),
            'xgboost Regressor': XGBRegressor( learning_rate=0.1,max_depth=5,n_estimators=300,  colsamplle_bytree=0.5)
            }

for i in range(len(list(models))):
            model = list(models.values())[i]
            model.fit(X_train,y_train) #train model

            #make prediction
            y_train_pred = model.predict(X_train)
            y_train_pred = model.predict(x_test)

            #evaluate train and test dataset
            model_train_mae,model_train_rmse,model_train_r2 = evaluate_model(y_train,X_train)
            model_test_mae,model_test_rmse,model_test_r2 = evaluate_model(y_test,X_test)

            print(list(models.keys())[i])
            print('Model performance for training set')
            print('Root Mean Squuared Error: {:,4f}'.format(model_train_r2)
            print('Mean Absolute Error: {:.4f}'.format(model_train_mae)
            print('R2 Score: {:.4f}'.format(model_train_rmse))

            print(f'---------------')

            print('Model performance for test set')
            print('Root Mean Squuared Error: {:,4f}'.format(model_test_r2)
            print('Mean Absolute Error: {:.4f}'.format(model_test_mae)
            print('R2 Score: {:.4f}'.format(model_test_rmse))
```

Unsupervised Machine Learning

Curse of Dimensionality

When training a model (Regression or Classification), increase in dimension (feature) will give better accuracy, but certain threshold point increase in dimension will start decrees in accuracy, this point of decrees in accuracy is called curse of dimensionality.

Feature Selection Vs Feature Extraction (Dimensionality Reduction)

Why Dimensionality Reduction

- Prevent curse of dimensionality
- Improve the performance of the mode
- Visualize the data (it brings n number of dimensions into 2D or 3D)

Feature Selection

Covariance is cov(x,y) = $\sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{n-1}$

**** if this value is +ve or -ve then we take these features for training, but if the value is ≈ 0, then we drop this feature for training***

Pearson Correlation Coefficients $P_{(x,y)} = \frac{cov\ (x,y)}{\sigma_x\ \sigma_y}$

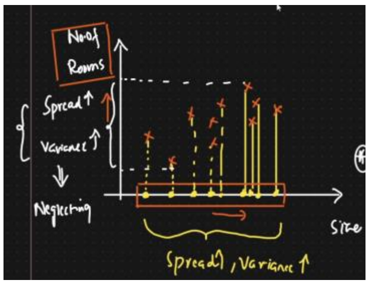
**** Helps to find the Strength and the Direction of the relationship, and the range $P_{(x,y)}$ between -1 to +1.***
**** The more towards the value of +1 the more +ve correlated***
**** The more towards the value of -1 the more -ve correlated***
**** The value ≈ 0 , then no relationship***

Feature Extraction

To reduce multiple feature into n number of features for model training (room_size,number_of_room => house_size)

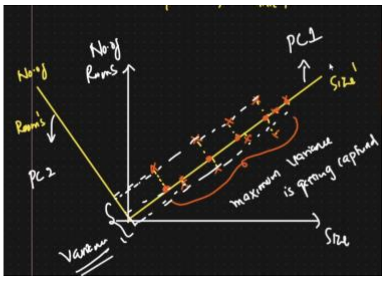
Principal Component Analysis (Dimensionality Reduction)

Principal Component Analysis (PCA) is a unsupervised ML algorithm, which helps to reduce the number of feature in the dataset.



- To reduce the features from data set, we have to process the data through Standard Normal Distribution $SD = \frac{x_i - \mu}{\sigma}$, were σ =1, μ=0 (use sklearn scalar sd)
- Rescale all the numerical value, so that all the value will be nearer to the PCA line. (use sklearn.preprocessing import StandardScalar)
- Apply PCA algorithm
- Apply ML algorithm

By converting from (exp.,2D into 1D), we loss of information (either one of the feature will be dropped), But inn PCA when convert from 2D to 3D there won't be much information lost



To select the maximum variance between vector we use ***Eigen Vector and Eigen Values***

- Covariance Matrix between features
- Eigen vectors and Eigen values will be found out from this covariance matrices

  $$linear\ transformation\ of\ metrix = A_{vector} = \lambda_{vector}$$

- for the Eigen vectors, where the Eigen value Is high (magnitude of the Eigen vector) this will capture the maximum variable features

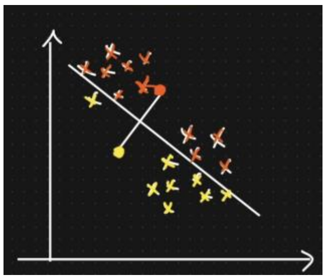
```
import pandas pd
import numpy as np
import matplotlib.pyplot as plt
import seabornn as sns
import plotly.express as pxx
import warnings
from skleaarn.datasets import
cancer_dataset == load_bread_cancer()

df = DataFrame(cancer_dataset['data'],colukn=cancer_dataset['feature_names'])

#standardization
from sklearn.preprocessing import StandardScalar
scaler = StandardScalar()

scaler.fit(df)
scaled_data = scaler.transform(df)

#applying PCA algorithms
from sklearn.decomposition import PCA
```

```
pca = PCA(n_componets=2)
data_pca = pca.fit_transform(scaled_data)
pcs.explained_variance_

pca = PCA(scaled_data)
pcs.explained_variance_

plt.figure(figsize=8,6)
plt.scatter(data_pca[:0],data_pca[:,1],cancer_dataset['target'],cmaps='plasma')
plt.xlabel('First principal component')
plt.ylabel('Second principal componet')
```

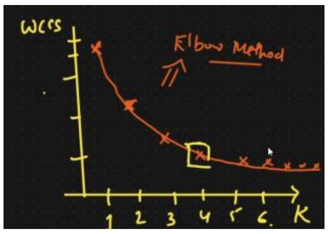
K Means Clustering

Points observed in K Means Clustering

1. Select K value (centroid value)
2. Initialize the centroid randomly
3. Select the group and find the mean of the group
4. The centroid will move into the group
5. Do the process from 2 to 4 until there is no movement between the groups
6. The distance between the centroid value is calculated by Euclidean distance method



Elbow method is used to select K value. When compute the K value (1 to 20) on $wcss$, for K=1, $wcss$ value will be high. When compute further (K=2, K=3, …) the $wcss$ value will be decreasing on certain computation for K value, $wcss$ will not change, that point of K value is optimized for to use in the model (K Means Clustering). The graph will look like Elbow.



Within cluster sum of square $wcss = \sum_{i=1}^{n}(c_i + x_i)^2$

Euclidian Distance $ED = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

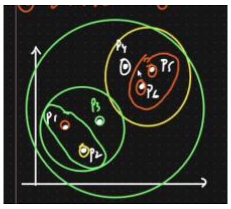
Manhattan Distance $MD = |(x_2 - x_1)| + |(y_2 - y_1)|$

```
import pandas pd
import numpy as np
import matplotlib.pyplot as plt
import seabornn as sns
import plotly.express as pxx
import warnings

Xx,y=make_blobs(n_samples=1000,centers=3,n_features=2
plt.scatter(X[:,0],X[:,1],c=y)

#standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

from sklearn.model_selection import train_test_split
X_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)

X_train = scaler.fit_transform(X_train)
x_test = scaler.transform(x_test)

from sklear.cluster import KMeans

#Elbow method to select K value
scss = []

for k in range(1,11):
        kmeans = KMeans(n_clusters=k,init = 'k-means++')
        kmeans.fit(X_train)
        wcss.append(kmeans.inertia_)

#plot elbow curve
plt.plot(range(1,11),wcss)
plt.xticks(range(1,11))
plt.xlabel('Number of Clluusters')
plt.ylabes('WCSS')
plt.show()

kmeans = KMeans(n_clusters=3,init = 'k-means++')

kmeans.fit_predict(X_train)
y_pred = kmeans.fit_predict(x_test)

plt.scatter(x_test[:,0],x_test[:,1],c=y_pred)

#validating the k value
#kneelocator
#sillhoutee scoring

#kneelocator
!pip install kneed
```

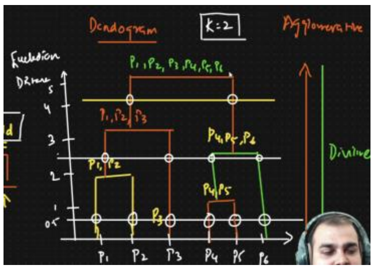
```
from kneed import KneeLocator
kl = KneeLocator(range(1,11),wcss,curve = 'convex',direction='decreasing')
kl.elbow

#sillhoutee scoring
from sklean.metrics i mport silhouette_score
silhouette_coefficients[]
for i in range(2,11):
        kmeans = Kmeans(n_cluster=k,init='k-means++')
        kmeans.fit(X_train)
        score = silhouette_score(X_train,kmeans.labels_)
        silhouette_coefficients.append(score)

plt.plot(range(2,11),silhouette_coefficients)
plt.xticks(range(2,11))
plt.xlabel('Number of Clluusters')
plt.ylabes('silhouette_coefficients')
plt.show()
```

Hierarchical Clustering

Agglomerative (combine)
Divisive (divide)
Steps:
For each point initially will consider it as a separate cluster
Find the nearest point an create new cluster
Keep on doing the same process until we get a single cluster



- Hierarchical Clustering is one of the Clustering methods by grouping closed points within a large cluster and plot on Dendogram.
- Dendogram is graphical representation of smallest, nearest grouping within big cluster.
- Dendogram helps to select to find number of clusters to be used in the model.



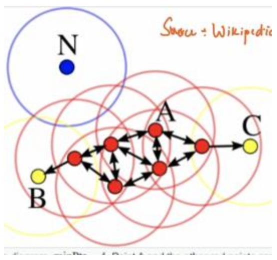
```
import pandas pd
import numpy as np
import matplotlib.pyplot as plt
import seabornn as sns
import plotly.express as pxx
import warnings

from sklearn import datasets

#import IRIS dataset

iris = dataset.load_iris()

iris_data = pd.DataFrame(iris)

from sklearn.preprocessing import StandardScalar

scaler = StandardScalar()
x_scaled = scaler.fit_trainform(iris_data)

#apply PCA

from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca_scalled = pca.fit_transform(x_scaled)

plt.scatter(pca_scalled[:,0],pca_scalled[:,1],c=iris.target)

#Agglommerative Cllustering
# to construct a dendogram
import scipy.cluster.hierarchy as sc
#plot the dendogram
plt.figure(figsize=10,7)
plt.title('Dengogram')

#create dendogram
sc.dendogram(sc.linkage(pca_scaled,method='ward'))
plt.title('Dendogram')
plt.slabel('sample Index')
plt.ylabel('Eculedian Distance')

from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=2,affinity='euclidian')
cluster.fit(pca_scaled)

cluster.label_
```

K Means Vs Hierarchal Clustering

- Data Set, Huge use K Means, Small use Hierarchal Clustering
- K Means Numerical data, Hierarchal Clustering other data type as well
- Visualization, in K Means getting centroid is difficult, Hierarchal Clustering just number of clusters

Density Based Spatial Clustering of Applications with Noise (DBSCAN)

Points to take in DBSCAN



1. Epsilon, radius of a referenced point
2. Minimal Points, set how many points should be present in the circular area of the referenced point (say point A), If Minimal point is 4, then 4 neighbor points should present in the circular area of the referenced point, say point A.
3. Core Points, If the circular area of the referenced point (A point) contains >= Minimal Points, then the referenced point is called Core Point
4. Border Points, If referenced point say point C is not contains the number of points (<) set from Minimal Points, but it contains at least one CORE Point (say A) within its circular area, then this point (C point) is called Border Point.
5. Noise Point, if referenced point say point D is not satisfy Minimal Points (< Minimal Point) and doesn't contain any Core Point within the circular area, then this point is called Noise Point. Noise Point is nothing but outlier.

Advantages of DBSCAN

- Is great at separating clusters of high density versus clusters of low density within a given datset
- Is great with handling outliers within the dataset

Disadvantages of DBSCAN

- Does not work well when dealing with clusters of varying densities, While DBSCAN is great at separating high density clusters from low density clusters, DBSCAN struggles with clusters of similar density.
- Struggles with high dimensionality data. I know, this entire article I have stated how DBSCAN is great at contorting the data into different dimensions and shapes. However, DBSCAN can only go so far, if given data with too many dimensions, DBSCAN suffers.

Step by Step to lean ML

1. Understand the math behind algorithms
2. How these algorithms behave w.r.t. numerical and categorical variables?
   a. Decision Tree uses different ways to split numerical and categorical predictors/variables.
3. How these algorithms work with Text Data?
   a. Stemming and Lemmatization
   b. Bag of words c. TF-DIF d. Word2Vec
4. For which scenario these algorithms are used?
   a. Regression - Linear regression
   b. Classification - Logistic regression, Naive Bayes Classifier
   c. Both - Decision Tree
5. Over-fitting and Under-fitting Conditions: -
   a. Hyper parameter tuning
   b. Decision Tree Pruning
6. Impact of Algorithm w.r.t Imbalanced Datasets and how do you fix that?
   a. Binary Classification Problem
   b. Feature Scaling # Up-sampling # Down-sampling
7. Impact of Outliers, how to treat them?
8. For which Algorithm, feature Scaling/Normalization is required w.r.t Datasets: -
   a. DT, Random Forest, XGBoost, Gradient Boosting, ADABoost - Not required
   b. Linear Regression, Logistic Regression - Required.

How to learn Statistics for Data Science

Basic Stats

1. Introduction to Basic Term
2. Variables
3. Random Variables
4. Population, Sample, Population Mean, Sample Mean
5. Population Distribution, Sample Distribution, and Sampling Distribution
6. Mean, Median, Mode
7. Range
8. Measure of Dispersion
9. Variance
10. Standard Deviation
11. Gaussian / Normal Distribution

Intermediate Stats

12. Standard Normal Distribution
13. Z score
14. Probability Density Function
15. Cumulative Distribution Functions
16. Hypothesis Testing
17. Many Different Plotting graphs

18. Kernel Density Estimations
19. Central Limit Theorem
20. Skewness of Data
21. Covariance
22. Pearson Correlation Coefficient
23. Spearman Rank Correlation

Advanced Stats

24. Q – Q Plot
25. Chebyshev's Inequality
26. Discrete and Continuous Distribution
27. Bernoulli and Binomial Distribution
28. Log Normal Distribution
29. Power Law Distribution
30. Box Cox Transform
31. Poisson Distribution
32. Application of Non-Gaussian Distribution

Outlier, Skewed and Impacts on Machine Learning Usecases

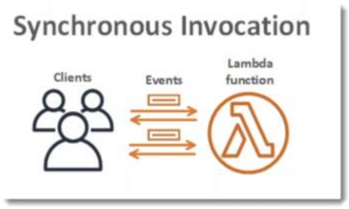
Which Machine Learning Models are Sensitive to Outliers

- Naivye Bayes Classifier – Not Sensitive to Outliers
- SVM – Not Sensitive to Outliers
- Linear Regression – Sensitive to Outliers
- Logistice Regression – Sensitive to Outliers
- Decision Tree Regressor or Classifier – Not Sensitive to Outliers
- Ensemble (RF,XGboost, GB) – Not Sensitive to Outliers
- KNN – Not Sensitive to Outliers
- Kmeans – Sensitive to Outliers
- Hierarchal – Sensitive to Outliers
- PCA – Sensitive to Outliers
- Neural Networks – Sensitive to Outliers

What is Synchronous Invocation

- with synchronous invocation you wait for the function to process the event and return a response

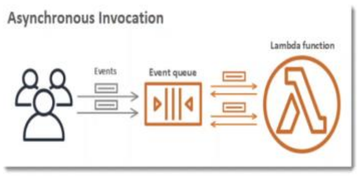
- Synchronous invocations are best suited for Machine Learning workflow



What is Asynchronous Invocation

- with asynchronous invocation, Lambda queues the event for processing, so you don't have to wait for a response from Lambda

- For asynchronous invocation, Lambda handles retries and can send invocation records to a destination


Asynchronous Invocation
Lambda function
Events
Event queue