EDA
&
DATA MANIPULATION

TEAM MEMBERS

ABHIJITH
ANN MARYA
ANSIL
ILLIYAS
MUHAMMED ANSHEER
SHERIN
YADU

1 EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is an approach to analyzing datasets to summarize their main characteristics, often with visual methods. EDA is used for seeing what the data can tell us before the modeling task.

1.1 Objective of Exploratory Data Analysis

The overall objective of exploratory data analysis is to obtain vital insights and hence usually includes the following sub-objectives:

- Identifying and removing data outliers
- Identifying trends in time and space
- Uncover patterns related to the target
- Creating hypotheses and testing them through experiments
- Identifying new sources of data

1.2 Why this is important

The primary purpose of EDA is to help deep look at the data set before making any assumptions, identifying obvious errors, gain a better understanding of the patterns within the dataset, figure out outliers and/or anomalous events, and last but not least, to find out the exciting relationships among the variables.

EDA is used to ensure the results the Data scientists are producing are valid and applicable to any desired goals. It helps the stakeholders to ensure that they are always asking the right questions. It also helps answer the questions about standard deviations, categorical variables, and confidence intervals. Finally, once EDA is complete and insights are drawn, its features can then be used for more sophisticated data analysis or modeling, including machine learning.

1.3 Different approaches to EDA

Depending on the number of columns we are analyzing we can divide EDA into three types.

1. **Univariate Analysis**
   In univariate analysis, we analyze or **deal with only one variable at a time**. The analysis of univariate data is thus the simplest form of analysis since the information deals with only one quantity that changes. It does not deal with causes or relationships and the main purpose of the analysis is to describe the data and find patterns that exist within it.

2. **Bi-Variate analysis**
   This type of **data involves two different variables**. The analysis of this type of data deals with causes and relationships and the analysis is done to find out the relationship between the two variables.

3. **Multivariate Analysis**
   When the data involves three or more variables, it is categorized under multivariate.

Depending on the type of analysis we can also subcategorize EDA into two parts.

1. **Non-graphical Analysis** – In non-graphical analysis, we analyze data using statistical tools like mean median or mode or skewness

2. **Graphical Analysis** – In graphical analysis, we use visualizations charts to visualize trends and patterns in the data

Univariate	Bivariate	Multivariate
It only summarizes single variable at a time.	It summarizes two variables	It summarizes more than 2 variables.
It does not deal with causes and relationships.	It deals with causes and relationships and analysis is done.	It does not deal with causes and relationships and analysis is done.
It does not contain any dependent variable.	It does contain only one dependent variable.	It is similar to bivariate but it contains more than 2 dependent variables.
The main purpose is to describe.	The main purpose is to explain.	The main purpose is to study the relationship among them.
The example of a univariate can be height.	The example of bivariate can be temperature and ice sales in summer vacation.	Example, Suppose an advertiser wants to compare the popularity of four advertisements on a website. Then their click rates could be measured for both men and women and relationships between variable can be examined

1.4 Univariate analysis

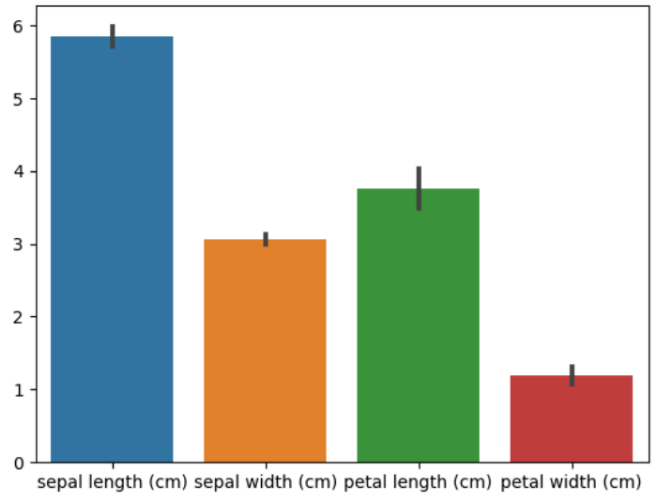
1.4.1 Graphical

1.4.1.1 Bar chart

A bar chart plots numeric values for **levels of a categorical feature as bars**. **Levels are plotted on one chart axis, and values are plotted on the other axis**. Each categorical value claims one bar, and the length of each bar corresponds to the bar's value. Bars are plotted on a common baseline to allow for easy comparison of values.

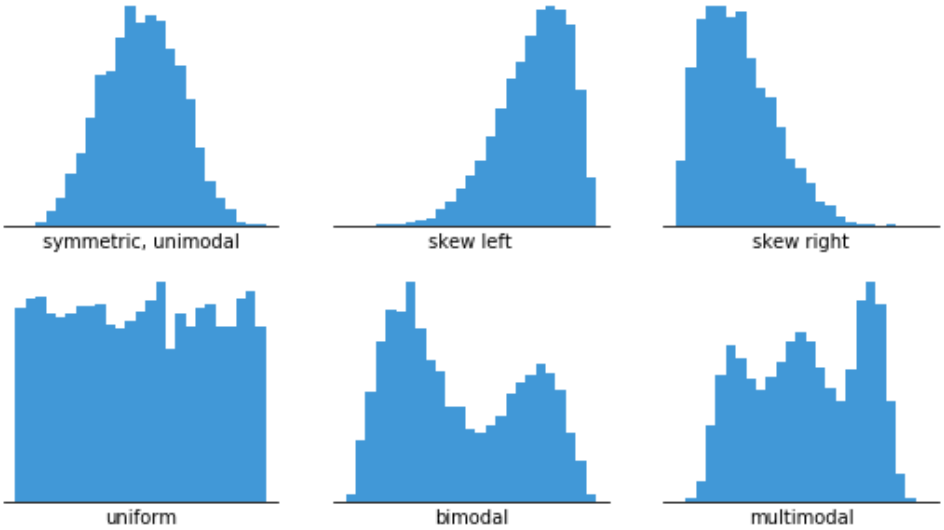
```
1 sns.barplot(irisd)
2
```

```
<Axes: >
```



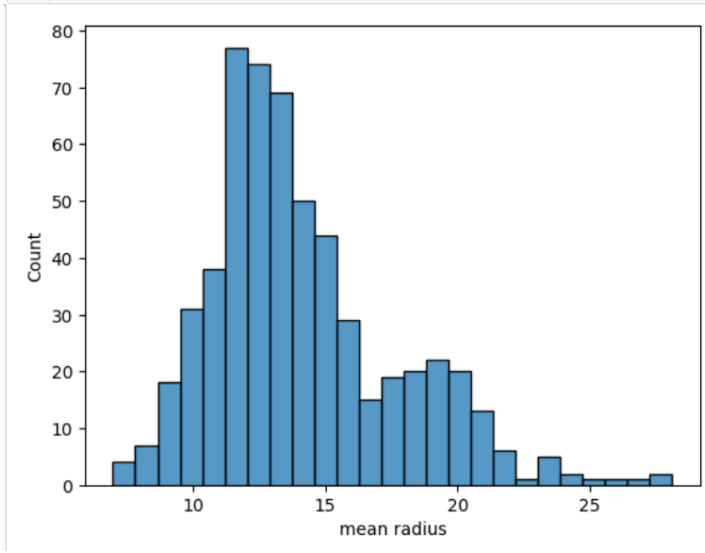
1.4.1.2 Histogram

It plots the distribution of a numeric variable's values as a series of bars. Each bar typically covers a range of numeric values called a bin or class; a bar's height indicates the frequency of data points with a value within the corresponding bin.



Histograms are good for showing general distributional features of dataset variables. You can see roughly where the peaks of the distribution are, whether the distribution is skewed or symmetric, and if there are any outliers.

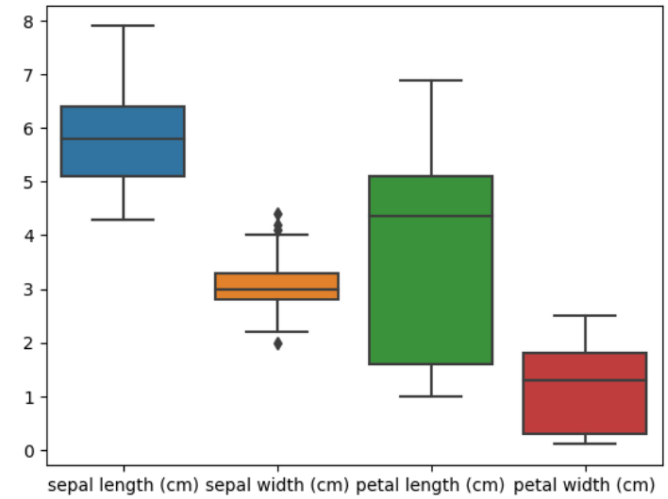
```
sns.histplot(canc['mean radius'], bins=25)
plt.show()
```



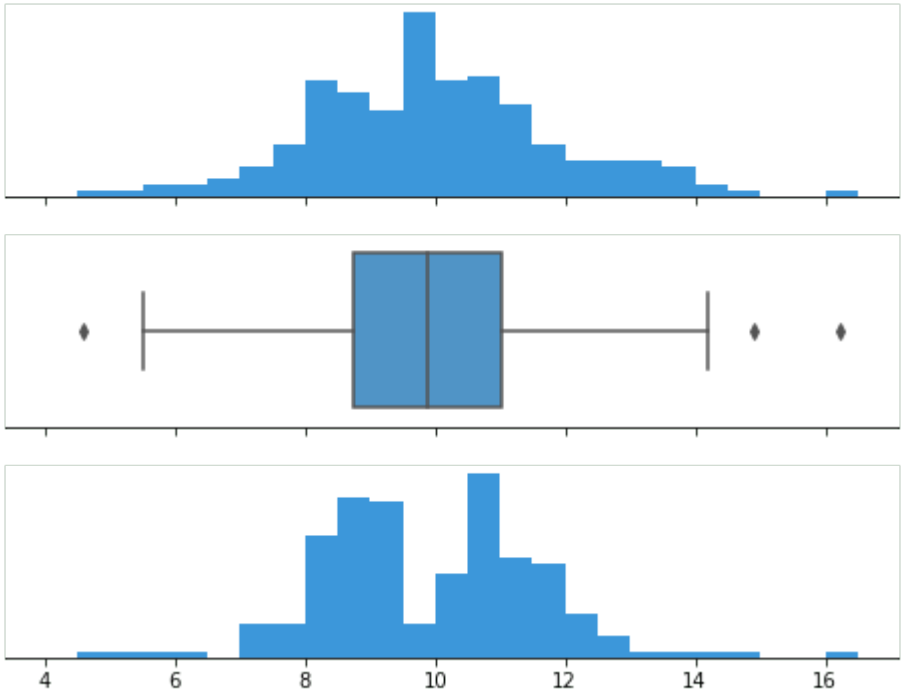
1.4.1.3 Box plot

Box plots are **used to show distributions of numeric data values**, especially when you want to compare them between multiple groups. They are built to provide high-level information at a glance, offering general **information about a group of data's symmetry, skew, variance, and outliers**. It is easy to see where the main bulk of the data is, and make that comparison between different groups.

```
sns.boxplot(irisd)
plt.show()
```



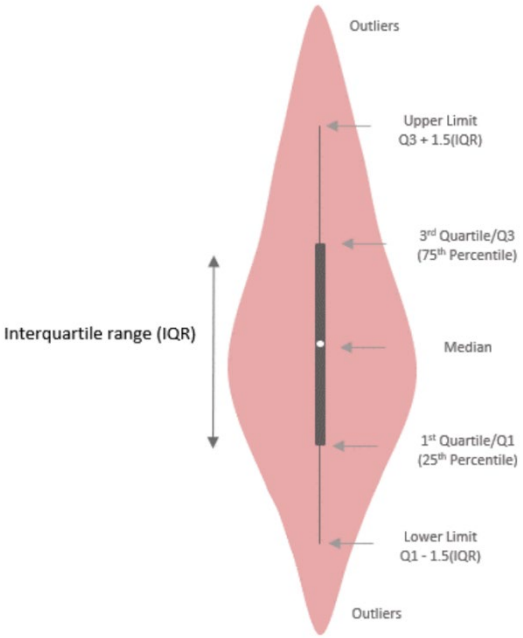
On the downside, a box plot's simplicity also sets limitations on the density of data that it can show. With a box plot, we miss out on the ability to observe the detailed shape of distribution, such as if there are oddities in a distribution's modality (number of 'humps' or peaks) and skew.



The datasets behind both **histograms** generate the same box plot in the center panel.

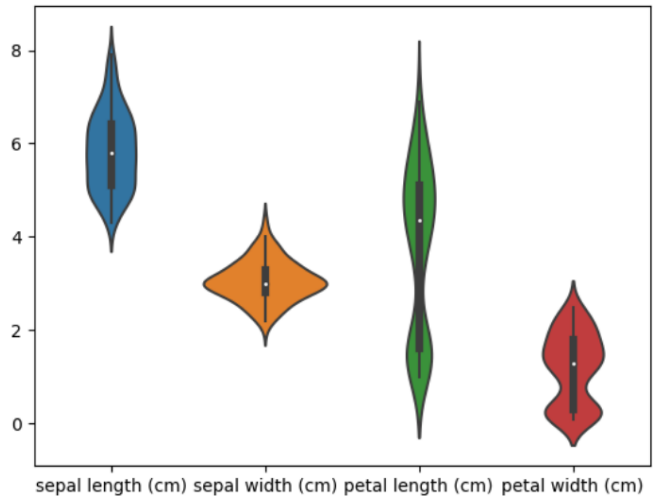
1.4.1.4 Violin plot

A violin plot depicts distributions of numeric data for one or more groups using density curves. **The width of each curve corresponds with the approximate frequency of data points in each region**. Densities are frequently accompanied by an overlaid chart type, such as box plot, to provide additional information.



Violin plots are used when you want to observe the distribution of numeric data, and are especially useful when you want to make a comparison of distributions between multiple groups. The peaks, valleys, and tails of each group's density curve can be compared to see where groups are similar or different. Additional elements, like box plot quartiles, are often added to a violin plot to provide additional ways of comparing groups

```
1 sns.violinplot(irisd)
2 plt.show()
```



1.4.1.5 Quantile-normal plots:

It's called the quantile-normal or QN plot or more generally the quantile-quantile or QQ plot. The quantile-quantile plot is a **graphical method for determining whether two samples of data came from the same population or not**. **It is a plot of the quantiles of the first data set against the quantiles of the second data set**. By a quantile, we mean the fraction (or percent) of points below the given value.

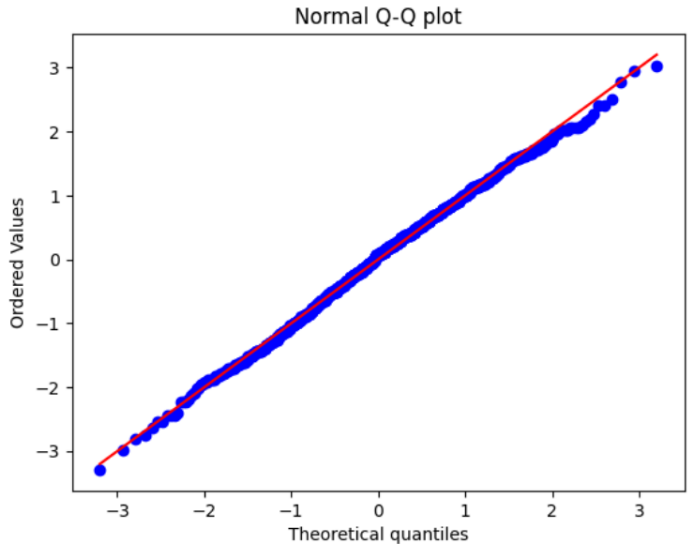
The Quantile-Quantile plot is used for the following purpose:

- Determine whether two samples are from the same population.
- Whether two samples have the same tail
- Whether two samples have the same distribution shape.
- Whether two samples have common location behavior.

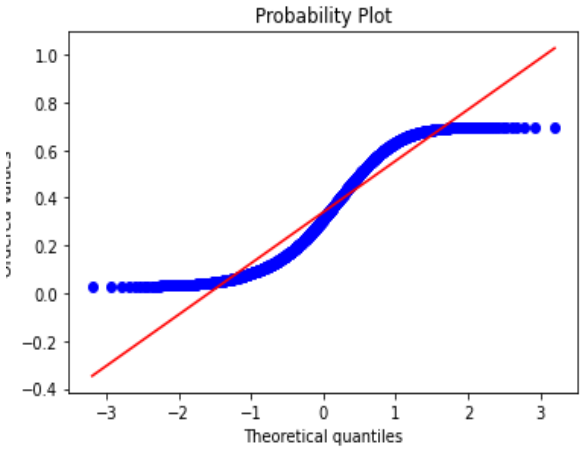
How to Draw Q-Q plot

- Collect the data for plotting the quantile-quantile plot.
- Sort the data in ascending or descending order.
- Draw a normal distribution curve.
- Find the z-value (cut-off point) for each segment.
- Plot the dataset values against the normalizing cut-off points.

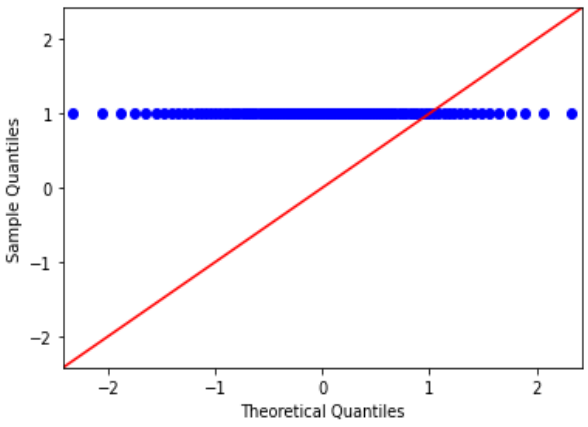
```python
import scipy.stats as stats
import numpy as np
import matplotlib.pyplot as plt

n = 2000
observation = np.random.binomial(n, 0.53, size=1000)/n

z = (observation-np.mean(observation))/np.std(observation)

stats.probplot(z, dist="norm", plot=plt)
plt.title("Normal Q-Q plot")
plt.show()

```



Types of Q-Q plots



For Left-tailed distribution



For the uniform distribution

1.4.2 Non graphical

1.4.2.1 Descriptive Statistics

In a nutshell, **descriptive statistics** aims to *describe* a chunk of raw data using summary statistics, graphs, and tables.

Descriptive statistics are useful because they **allow you to understand a group of data** much more quickly and easily compared to just staring at rows and rows of raw data values.

1.4.2.1.1 Summary Statistics

The most common way to perform univariate analysis is to describe a variable using summary statistics.

There are two popular types of summary statistics:

- **Measures of central tendency:** these numbers describe where the center of a dataset is located. Examples include the *mean* and the *median*.
- **Measures of dispersion:** these numbers describe how spread out the values are in the dataset. Examples include the *range*, *interquartile range*, *standard deviation*, and *variance*.

```
1 df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

1.4.2.1.2 Frequency Distributions

Another way to perform univariate analysis is to create a frequency distribution, which describes how often different values occur in a dataset.

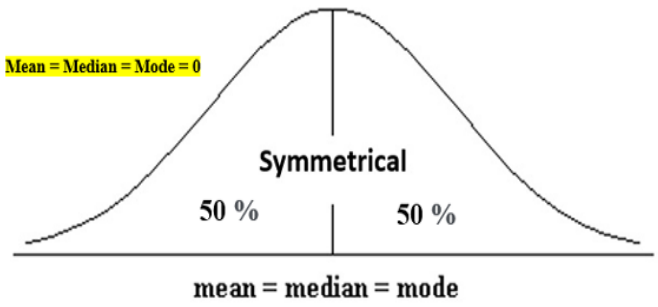
Number of Pets	Frequency	Relative Frequency	
1	150	37.5%	← 150/400 = 37.5%
2	90	22.5%	← 90/400 = 22.5%
3	110	27.5%	← 110/400 = 27.5%
4	30	7.5%	← 30/400 = 7.5%
5	20	5.0%	← 20/400 = 5.0%

.

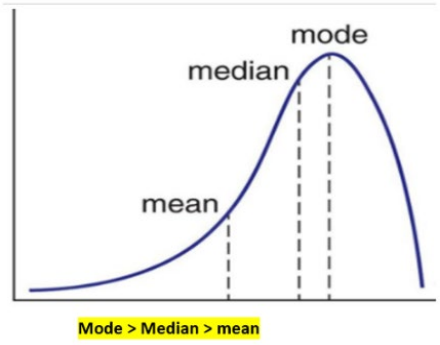
1.4.2.1.3 Skewness and kurtosis

1.4.2.1.3.1 Skewness

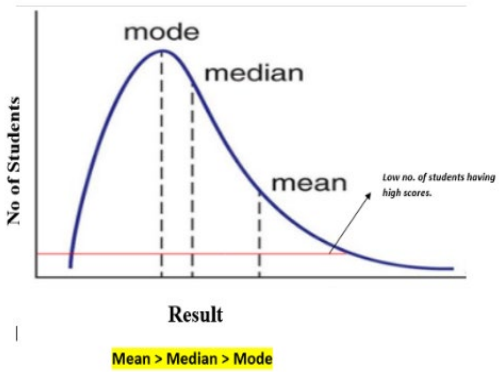
Skewness is typically used to **describe the shape of a univariate distribution**. A distribution can be positively skewed, negatively skewed, or approximately symmetric (i.e., no skewness).



Normal/ No skew



Negative/left skew



Positive/right skew

Skewness is measured using a skewness coefficient. One common way to calculate skewness is to use Pearson's first coefficient of skewness, given by:

$$Skewness = \frac{3(Mean - Median)}{Standard\ Deviation}$$

Here's how the skewness value can help in EDA and machine learning:

- **Identifying data distribution**: Skewness allows us to understand the shape of the data distribution. It helps to recognize whether the data is skewed to the left or right, or if it is symmetric.
- **Data preprocessing**: Skewed data can sometimes impact the performance of machine learning algorithms, particularly those sensitive to the distribution of data (e.g., linear regression). In such cases, data transformation techniques like log transformation, square root transformation, or box-cox transformation can be applied to make the data more symmetric.
- **Decision-making**: Understanding skewness helps in making informed decisions about which statistical tests and models are suitable for the data.

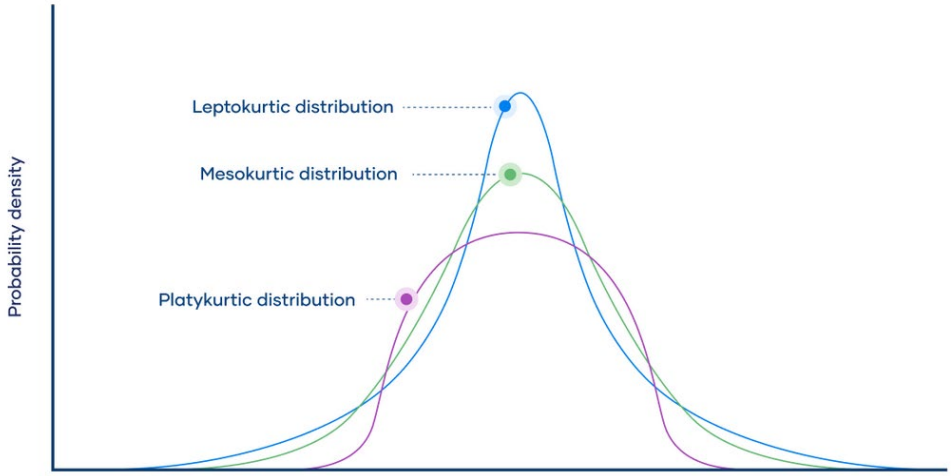
1.4.2.1.3.2 Kurtosis

Kurtosis is another statistical measure used in Exploratory Data Analysis (EDA) to **describe the shape of a probability distribution of a dataset**. **It provides information about the heaviness of the tails of the distribution and whether it has more outliers or extreme values compared to a normal distribution.**

In the context of machine learning and data analysis, understanding kurtosis can be beneficial in several ways:

- **Characterizing Data Distributions**: Kurtosis helps in identifying the shape of the distribution of a univariate dataset. It allows us to distinguish between various types of distributions, such as normal, leptokurtic (heavy-tailed), and platykurtic (light-tailed).
- **Assessing Outliers**: A high kurtosis value indicates heavy tails in the distribution, which means the dataset has more outliers or extreme values compared to a normal distribution. This information is useful for anomaly detection and outlier analysis.
- **Modeling Selection**: Some statistical models assume a specific distribution of the data, often assuming normality. In cases where the data has high kurtosis, it may

be necessary to consider different modeling techniques or apply data transformations to achieve better results.



Types of Kurtosis

- **Leptokurtic**
  Leptokurtic has very long and thick tails, which means there are **more chances of outliers**. Positive values of kurtosis indicate that distribution is peaked and possesses thick tails. Extremely positive kurtosis indicates a distribution where more numbers are located in the tails of the distribution instead of around the mean.
- **Platykurtic**
  Platykurtic having a thin tail and stretched around the center means most data points are present in high proximity to the mean. A platykurtic distribution is flatter (less peaked) when compared with the normal distribution.
- **Mesokurtic** (Kurtosis = 3)
  Mesokurtic is the same as the normal distribution, which means kurtosis is near 0. In Mesokurtic, distributions are moderate in breadth, and curves are a medium peaked height.

1.4.2.2 Inferential Statistics

In a nutshell, inferential statistics **uses a small sample of data** to draw *inferences* about the larger population that the sample came from.

1.4.2.2.1 Hypothesis Testing

The process of hypothesis testing is to draw inferences or some conclusion about the overall population or data by conducting some statistical tests on a sample like T-test, Z-test, Chi-square test etc.

It's important to note that hypothesis testing is just one aspect of EDA, and it should be used judiciously. Proper interpretation of results and understanding the limitations of hypothesis testing are crucial to drawing valid conclusions from the data.

Remember that the results of hypothesis testing are only as reliable as the assumptions underlying the statistical tests. Always consider the context of your data and the assumptions made before drawing any final conclusions from the hypothesis tests.

Some examples

Univariate Hypothesis Tests:

- One-Sample t-test: Tests whether the mean of a single sample differs significantly from a known or hypothesized population mean.
- Paired t-test: Tests whether the mean of the differences between two related samples differs significantly from zero.
- Independent Samples t-test: Tests whether the means of two independent samples differ significantly from each other.
- One-Way ANOVA: Tests whether the means of multiple groups (more than two) are significantly different from each other.
- Chi-Square Test for Goodness of Fit: Tests whether observed categorical data fits an expected distribution.
- Runs Test: Tests whether a sequence of binary outcomes is random.

Bivariate Hypothesis Tests:

- Pearson Correlation Test: Tests whether there is a linear relationship between two continuous variables.
- Spearman Rank Correlation Test: Non-parametric test of the strength of association between two ranked variables.
- Chi-Square Test of Independence: Tests whether two categorical variables are independent of each other.
- Fisher's Exact Test: Used for small sample sizes to test the independence of two categorical variables.

Multivariate Hypothesis Tests:

- Hotelling's T-Squared Test: Multivariate extension of the t-test, used to compare means of two multivariate samples.
- MANOVA (Multivariate Analysis of Variance): Tests whether there are differences in multiple dependent variables between two or more groups.

- Principal Component Analysis (PCA) Significance Test: Tests whether the principal components are significantly different from random noise.
- Canonical Correlation Analysis (CCA) Test: Tests the relationship between two sets of multivariate variables.

1.4.2.2.2 Confidence interval

In simple terms, Confidence Interval is a **range where we are certain that true value exists**.

In EDA, confidence intervals help you understand the range within which the true population parameter (mean, proportion, etc.) is likely to lie based on the observed sample data. It provides a measure of the precision of your estimates.

In machine learning, confidence intervals can be used to assess the performance of models and compare different models or algorithms.

- **EDA with Confidence Intervals**: In EDA, you often work with a sample of data to understand the underlying population distribution. When calculating summary statistics (e.g., mean, median, variance) from your sample, you can't be entirely certain that these statistics perfectly represent the population values. Confidence intervals offer a way to express the uncertainty around these estimates.

  For example, if you want to estimate the average age of customers in a certain region based on a sample of data, you can compute a confidence interval around the sample mean. The confidence interval will give you a range within which the true average age of all customers in that region is likely to fall.

- **Model Evaluation in Machine Learning**: In machine learning, you often evaluate model performance using metrics like accuracy, precision, recall, or F1 score. When comparing models or reporting performance to stakeholders, it's essential to consider the uncertainty associated with these metrics, especially when working with limited data.

  Confidence intervals for performance metrics provide a way to quantify the variability of model performance across different random samples or data splits. It helps you avoid drawing definitive conclusions based on a single performance score and allows you to compare models more effectively.

To compute confidence intervals for model performance, you can use techniques like bootstrapping or cross-validation

Keep in mind that confidence intervals are related to sample size, and larger sample sizes generally result in narrower intervals, providing more precise estimates.

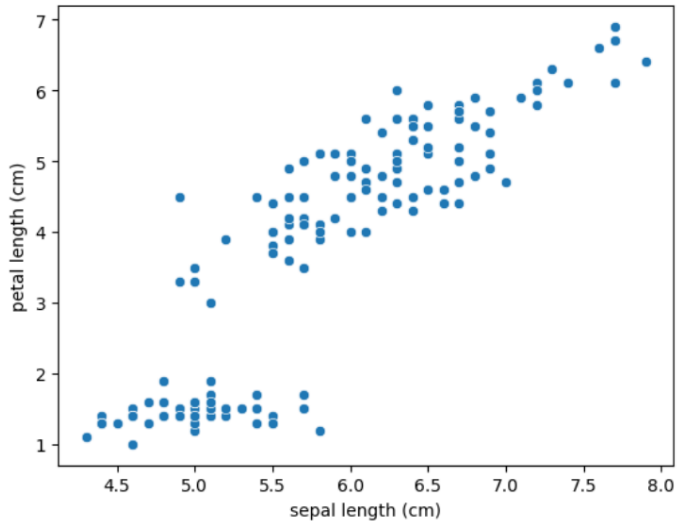
1.5 Bivariate analysis

1.5.1 Graphical

1.5.1.1 Scatter plot

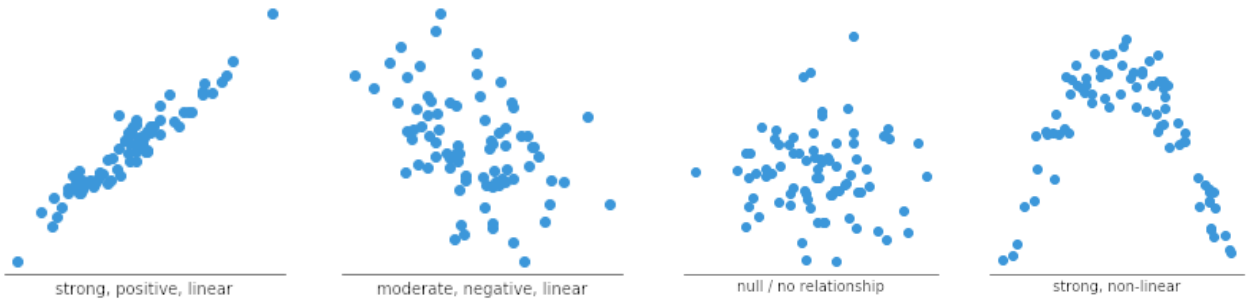
A scatter plot (aka scatter chart, scatter graph) uses **dots to represent values for two different numeric variables**. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.

```
1 sns.scatterplot(x=irisd['sepal length (cm)'],y=irisd['petal length (cm)'])
2 plt.show()
```



Identification of correlational relationships are common with scatter plots. In these cases, we want to know, if we were given a particular horizontal value, what a good prediction would be for the vertical value.

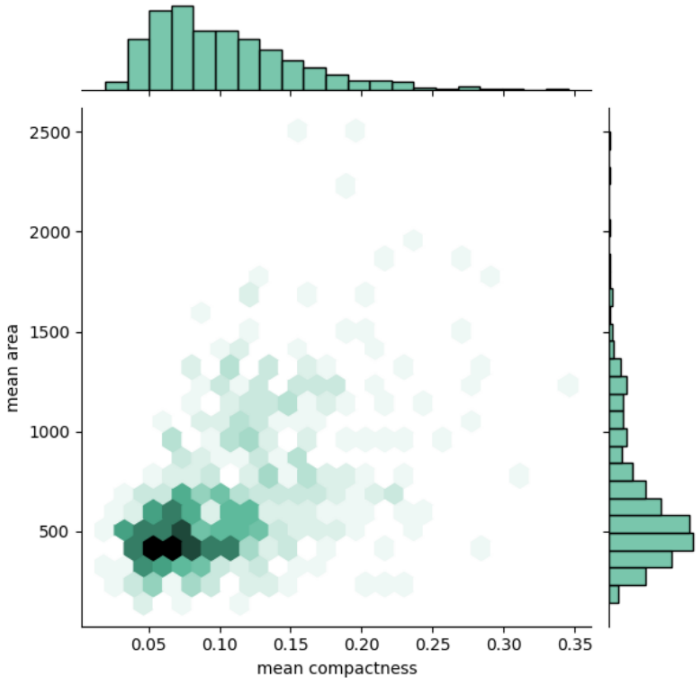
Relationships between variables can be described in many ways: positive or negative, strong or weak, linear or nonlinear.



1.5.1.2 Hexplot

Splits the plotting window into several hexbins and then the number of observations which fall into each bin corresponds with a **color to indicate density**.

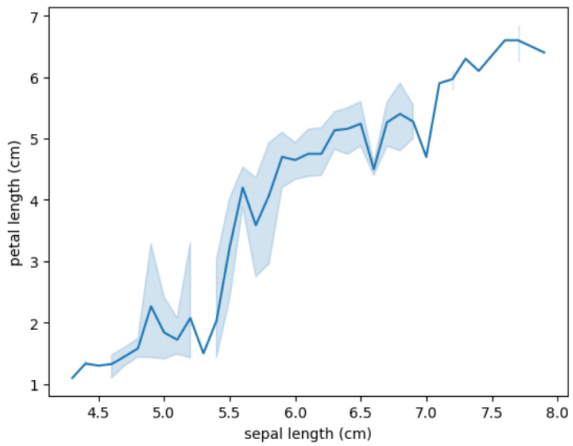
```
sns.jointplot(x=canc['mean compactness'],y=canc['mean area'], kind='hex', color="#4CB391")
plt.show()
```



1.5.1.3 Line chart

A line chart, also referred to as a line graph or a line plot, **connects a series of data points using a line**. This chart type presents sequential values to help you identify trends. Most of the time, the x-axis (horizontal axis) represents a sequential progression of values.

```
sns.lineplot(irisd, x=irisd['sepal length (cm)'], y=irisd['petal length (cm)'])
plt.show()
```



1.5.2 Non graphical

1.5.2.1 Correlation

Correlation is a statistical measure that **shows how strong and in what direction two variables are linked.** A positive correlation means that when one variable goes up, so does the other. A negative correlation shows that when one variable goes up, the other one goes down.

```
1 df.corr()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000

1.5.2.2 Regression analysis

Regression analysis is a common technique used in bivariate analysis to model the **relationship between a dependent variable (also known as the target or outcome variable) and an independent variable (also known as the predictor or feature variable).**

The goal of regression analysis is **to find the best-fitting line (in the case of simple linear regression) or plane/hyperplane (in the case of multiple linear regression) that represents the relationship between the variables.**

1.5.2.3 Chi-square test

The chi-square test is another statistical method used in Exploratory Data Analysis (EDA), **particularly in the context of categorical data. It is employed to determine if there is a significant association between two categorical variables**. In other words, it helps to assess whether the observed distribution of frequencies in a contingency table significantly differs from the expected distribution, assuming that the two variables are independent.

However, it's essential to note that the chi-square test assesses association, not causation, between variables.

1.5.2.4 T-test

A t-test is a statistical test **used to determine if there is a significant difference between the means of two groups in a sample**. It is commonly used when the sample size is small, and the population standard deviation is unknown. The t-test assesses whether the means of the two groups are significantly different from each other based on the t-statistic and the associated p-value.

1.5.2.5 Z-test

A z-test is another statistical test **used to determine if there is a significant difference between the means of two groups in a sample**. It is **similar to the t-test but is used when the sample size is large, and the population standard deviation is known**. In practice, the z-test is often used when the sample size is sufficiently large, and the population standard deviation can be estimated from prior knowledge or historical data.

1.5.2.6 ANOVA (Analysis of Variance)

ANOVA, which stands for Analysis of Variance, is a statistical method **used to compare the means of two or more groups** or treatments to determine if there are any significant differences between them. **It is an extension of the t-test, which can only compare two groups, to scenarios with multiple groups**.

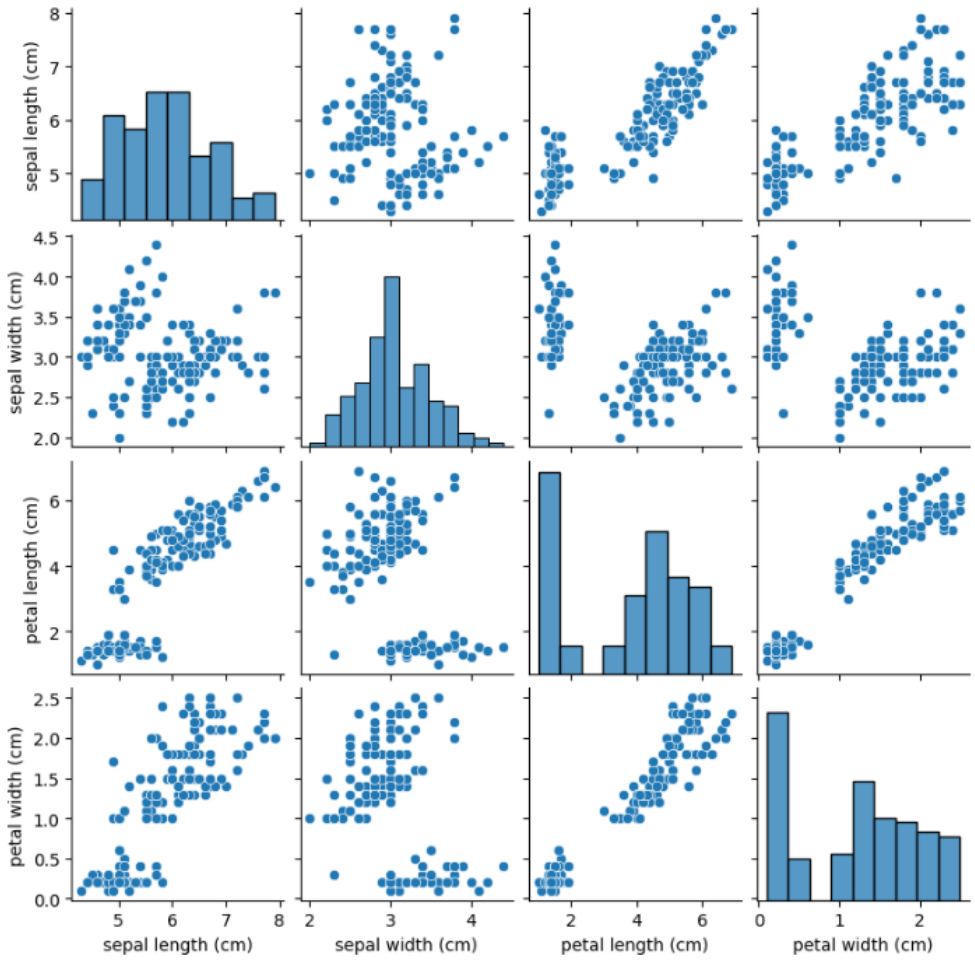
1.6 Multivariate analysis

1.6.1 Graphical

1.6.1.1 Multivariate Chart

A Multivariate chart is a type of control **chart used to monitor two or more interrelated process variables**. This is beneficial in situations such as process control, where engineers are likely to benefit from using multivariate charts. These charts allow monitoring multiple parameters together in a single chart. A notable advantage of using multivariate charts is that they help minimize the total number of control charts for organizational processes. Pair plots generated using the Seaborn library are a good example of multivariate charts as they help visualize the relationships between all numerical variables in the entire dataset at once.

```
1 sns.pairplot(irisd,size=2)
2 plt.show()
```

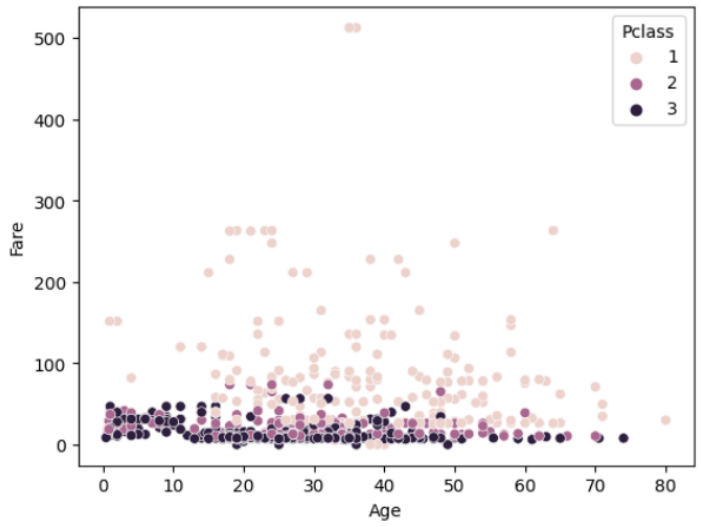


1.6.1.2 Bubble Chart

Bubble charts **scatter plots that display multiple circles (bubbles) in a two-dimensional plot.** These are used to **assess the relationships between three or more numeric variables**. In a bubble chart, every single dot corresponds to one data point, and the values of the variables for each point are indicated by different positions such as horizontal, vertical, dot size, and dot colors.

```
1 sns.scatterplot(data=titanic,x='Age',y = 'Fare', hue='Pclass')
```

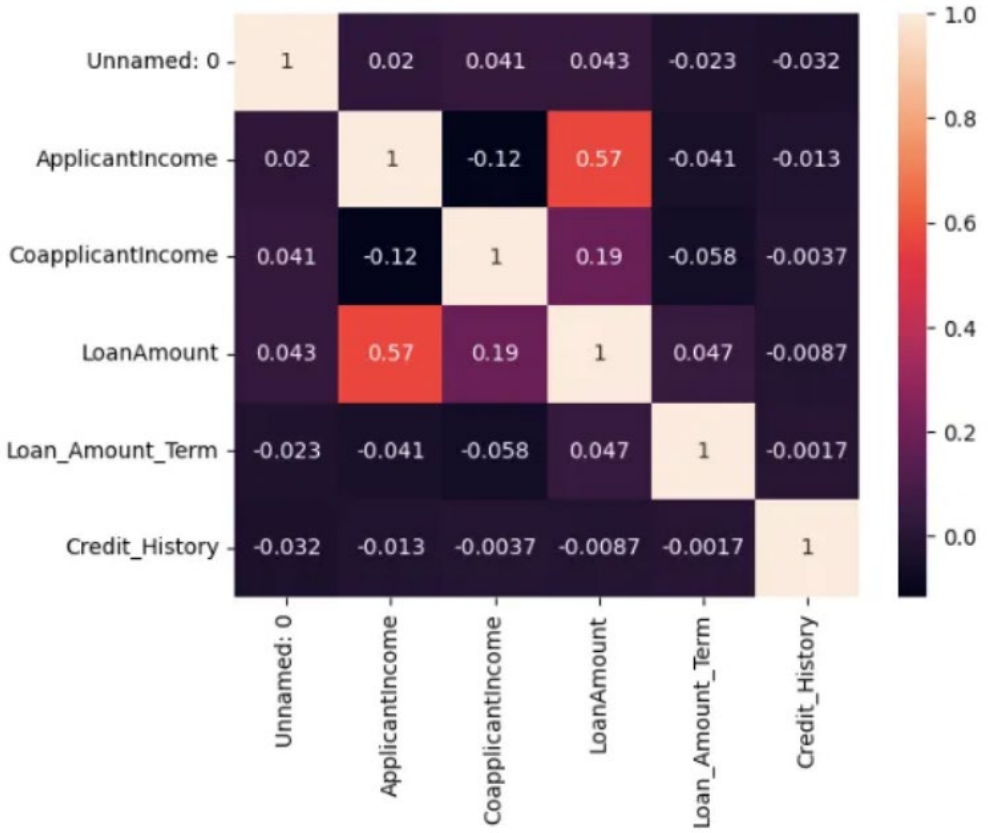
```
<Axes: xlabel='Age', ylabel='Fare'>
```



1.6.1.3 Heat Map

A heat map is a **colored graphical representation of multivariate data structured as a matrix of columns and rows**. The heat map **transforms the correlation matrix into color coding and represents these coefficients to visualize the strength of correlation among variables**. It assists in finding the best features suitable for building accurate Machine Learning models. Also help in finding columns to be dropped as part of reducing dependency between features.

```
sns.heatmap(data.corr(), annot=True)
```



1.6.2 Non-Graphical

1.6.2.1 Additive Tree.

An additive tree is a useful tool to understand the relationships between multiple variables in a dataset. Additive trees are also known as "exploratory regression trees" or "MARS" (Multivariate Adaptive Regression Splines).

An additive tree is a **non-linear regression model that represents the target variable as a sum of simpler sub-models, typically piecewise linear functions, which are defined over different regions of the input space. Each sub-model captures the relationship between the target variable and a specific set of predictor variables within a particular region**.

Where we can use additive trees in EDA:

- **Identifying Important Features**: Additive trees can help identify the most significant features in the dataset, allowing analysts to focus on the most relevant variables for further investigation.
- **Segmentation**: Decision trees can be used to segment the data into distinct groups or clusters based on feature combinations, aiding in the discovery of underlying patterns or subpopulations.
- **Relationship Exploration**: Additive trees can reveal non-linear relationships between variables, capturing interactions and dependencies that might not be evident through traditional linear analysis.
- **Handling Mixed Data**: If the dataset contains a mix of numerical and categorical variables, decision trees can accommodate them without the need for extensive data preprocessing.
- **Outlier Detection**: By constructing a decision tree to represent the majority behavior of the data, outliers can be identified as instances that deviate from the expected patterns.
- **Data Imputation**: Additive trees can be used for data imputation by predicting missing values based on available features.

Where we may not use additive trees in EDA:

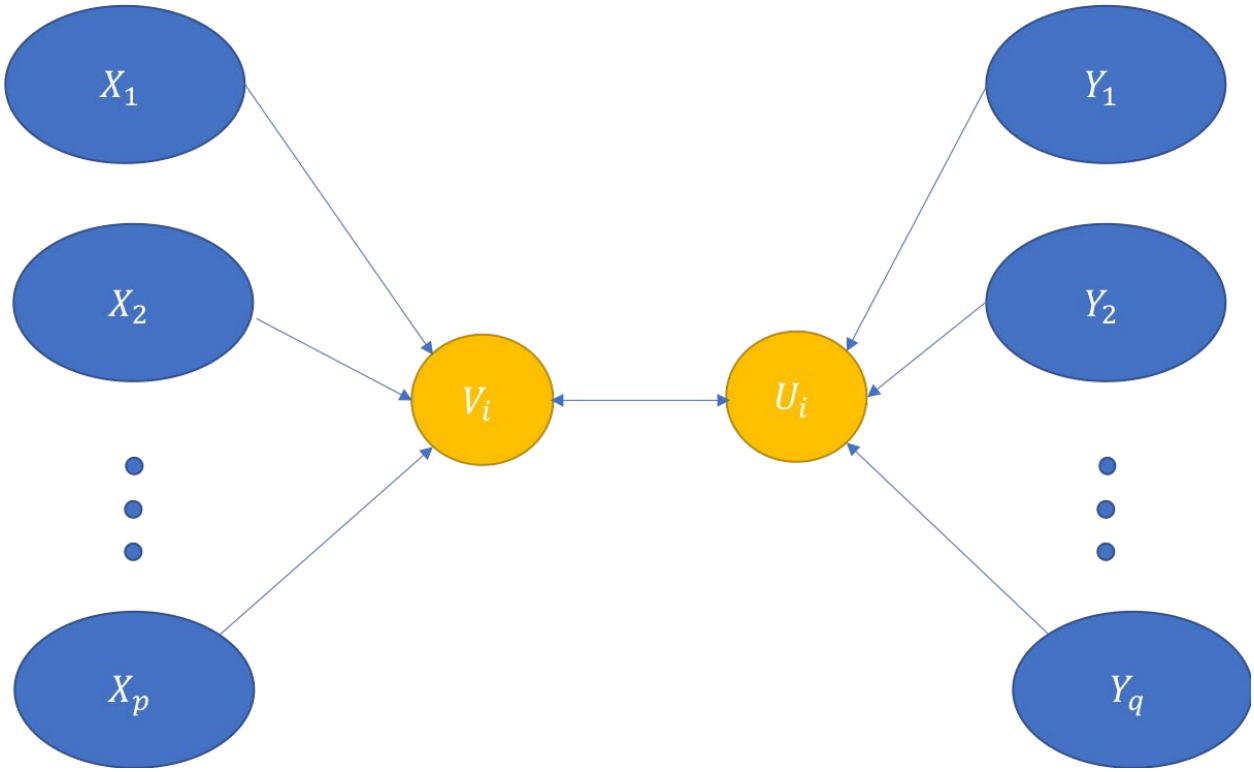
- **Data Exploration with Many Features**: If the dataset has a large number of features, the resulting decision tree can become complex and hard to interpret. In such cases, simpler visualization techniques or dimensionality reduction methods may be more suitable.
- **Highly Correlated Features**: When the dataset contains highly correlated features, decision trees might be less effective at capturing unique contributions from each feature, potentially leading to biased results.
- **Continuous Data with Few Unique Values**: In situations where the continuous data has very few unique values, decision trees may not be the best choice as they might not capture the underlying distribution well.
- **Time Series Analysis**: For time series data, traditional decision trees might not be the most appropriate choice, as they do not inherently account for the temporal ordering of observations. Time series-specific techniques like ARIMA or SARIMA models could be more suitable.
- **Interpolation or Extrapolation**: Additive trees are not well-suited for interpolation or extrapolation beyond the range of the observed data. For such tasks, other modeling techniques may be more appropriate.

1.6.2.2 Canonical Correlation Analysis.

Canonical correlation analysis (CCA)is a statistical technique to derive the relationship between two sets of variables. **One way to understand the CCA, is using the concept**

of multiple regression. In multiple regression, the relationship between one single dependent variable and a set of independent variables are investigated. In CCA, we extend the multiple regression concept to more than one dependent variable.

Furthermore, it is a popular statistical technique and is widely used in many areas of social science, psychological research, and marketing analytics. Unlike a regression analysis, researchers can monitor the relationship between many dependent and independent variables.



two sets of variables connected through canonical variables

1.6.2.3 Cluster Analysis.

Cluster analysis, also known as clustering, is a method of data mining that **groups similar data points together.** The goal of cluster analysis is to divide a dataset into groups (or clusters) such that the data points within each group are more similar to each other than to data points in other groups.

Some Clustering Methods:

1. Partitioning Method
2. Hierarchical Method
3. Density-based Method

4. Grid-Based Method
5. Model-Based Method
6. Constraint-based Method

1.6.2.4 Correspondence Analysis / Multiple Correspondence Analysis

Correspondence analysis is an statistical technique that **can show us the relationships between the categories within two variables, based on data given in a contingency table**. It uses a graph that plots data, visually showing the outcome of two or more data points.

Correspondence analysis versus multiple correspondence analysis: which to use and when?

Multiple correspondence analysis is a technique for analyzing categorical variables. It is essentially a form of factor analysis for categorical data. You should use it **when you want a general understanding of how categorical variables are related.**

Correspondence analysis is a technique for summarizing relativities on tables. As tables are ubiquitous in data analysis, it is a technique that can be used widely.

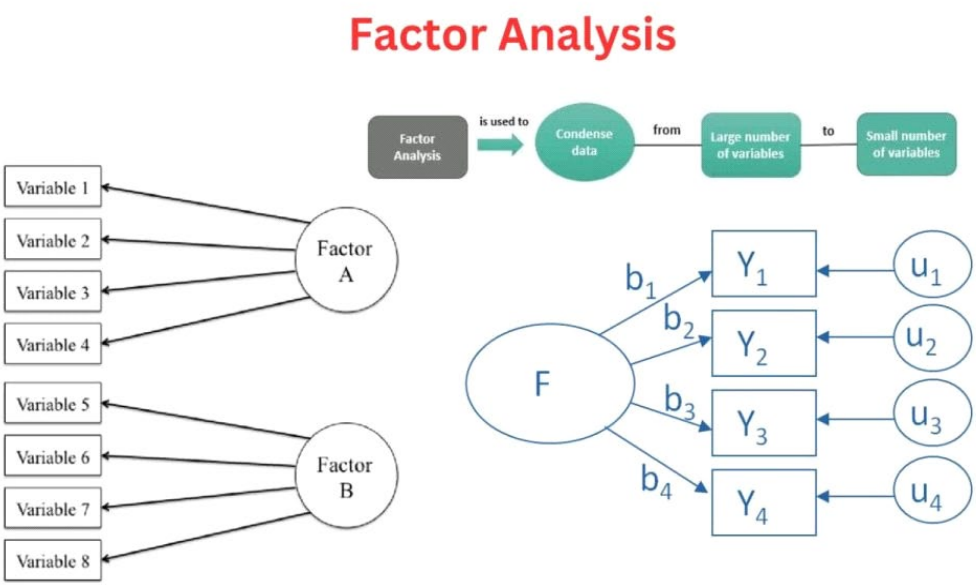
Both techniques give the same answer when you have two variables.

The reason for the word "multiple" is that multiple correspondence can be provided to a table that has more than two dimensions (e.g., a cube), whereas correspondence analysis requires as an input a table with only two dimensions.

1.6.2.5 Factor Analysis.

Factor Analysis (FA) is an exploratory data analysis method **used to search influential underlying factors or latent variables from a set of observed variables. It helps in data interpretations by reducing the number of variables.** It extracts maximum common variance from all variables and puts them into a common score.

Factor analysis is widely utilized in market research, advertising, psychology, finance, and operation research. Market researchers use factor analysis to identify price-sensitive customers, identify brand features that influence consumer choice, and helps in understanding channel selection criteria for the distribution channel.



What is a factor?

A factor is a latent variable which describes the association among the number of observed variables. The maximum number of factors are equal to a number of observed variables. Every factor explains a certain variance in observed variables. The factors with the lowest amount of variance were dropped. Factors are also known as latent variables or hidden variables or unobserved variables or Hypothetical variables.

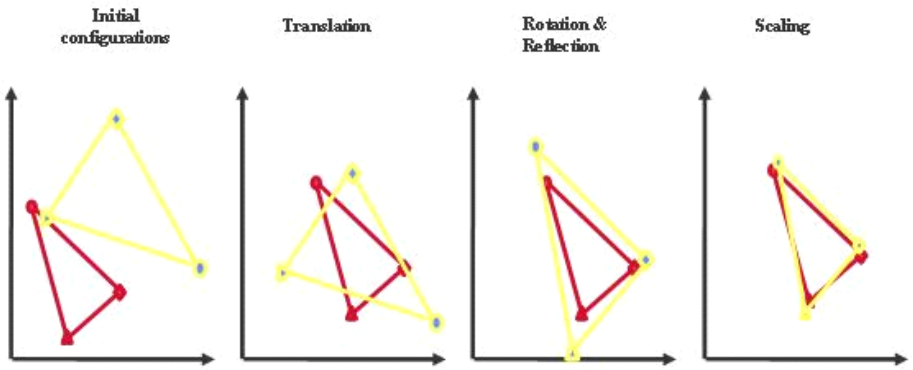
1.6.2.6 Generalized Procrustean Analysis.

Generalized Procrustean Analysis (GPA) is a technique used in exploratory data analysis (EDA) and machine learning to compare and align multiple datasets in a common space. It is especially useful when dealing with data that might be recorded under different conditions or with different measurement scales.

In simple words, GPA helps to find a common reference frame for multiple datasets so that they can be directly compared. It aims to remove any variations caused by differences in measurement scales, rotations, or translations among the datasets.

Keep in mind that the applicability of GPA depends on the specific nature of the datasets and the objectives of the EDA. It might not be suitable for all EDA scenarios, and its usage should be considered based on the specific context of the data analysis task.

Python has libraries that can help you perform Generalized Procrustean Analysis, such as the SciPy library. You can use the scipy.spatial.procrustes function for this purpose.



Shape Comparison

1.6.2.7 Homogeneity of Covariance

Homogeneity of covariance is an assumption made in various statistical methods and machine learning algorithms when dealing with multivariate data and multiple groups or classes.

It assumes that the variance-covariance matrix (the matrix that contains both variances and covariances) is the same across all groups or classes. In other words, it assumes that the data points from different groups share a common covariance structure.

Violating the homogeneity of covariance assumption can lead to biased parameter estimates and unreliable results in methods that assume this assumption, such as linear discriminant analysis (LDA) or quadratic discriminant analysis (QDA).

1.6.2.8 Independent Component Analysis (ICA)

ICA helps in uncovering latent structures in data by revealing underlying patterns and dependencies among variables, which may not be evident through traditional data analysis techniques.

The fundamental idea behind ICA is to transform the observed mixed data into statistically independent components. It assumes that the observed data is a linear combination of the independent sources, and the goal is to estimate the mixing matrix and the independent components.

In the context of machine learning, ICA can be employed as a **preprocessing step** to enhance the quality of input data for downstream tasks, or as a **method for feature**

extraction, which can be useful when dealing with highly correlated features or when the dataset has a large number of variables.

ICA is related to other dimensionality reduction techniques like Principal Component Analysis (PCA) and Factor Analysis (FA). However, ICA focuses on finding statistically independent components while PCA seeks orthogonal components that capture the most variance, and FA assumes that observed variables are linear combinations of a few underlying latent factors plus noise.

Keep in mind that while ICA is a powerful tool, it might not always be appropriate for every dataset. Its effectiveness depends on the underlying assumptions and the nature of the data being analyzed

1.6.2.9 MANOVA

MANOVA (multivariate analysis of variance): It is a type of multivariate analysis used to analyze **data that involves more than one dependent variable at a time**. MANOVA allows us to test hypotheses regarding the effect of one or more independent variables on two or more dependent variables. The obvious difference between ANOVA and the "Multivariate Analysis of Variance" (MANOVA) is the "M", which stands for multivariate. **In basic terms, MANOVA is an ANOVA with two or more continuous response variables**. Like ANOVA, MANOVA has both the one-way flavor and a two-way flavor. The number of factor variables involved distinguishes the one-way MANOVA from a two-way MANOVA.

1.6.2.10 Multidimensional Scaling.

Multidimensional scaling (MDS) is a dimensionality reduction technique that is used to **project high-dimensional data onto a lower-dimensional space while preserving the pairwise distances between the data points as much as possible**. MDS is based on the concept of distance and aims to find a projection of the data that minimizes the differences between the distances in the original space and the distances in the lower-dimensional space.

1.6.2.11 Multiple Regression Analysis.

Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. That is Multiple Linear Regression **models the linear relationship between a single dependent continuous variable and more than one independent variable**.

1.6.2.12 Partial Least Square Regression.

Partial Least Squares regression **(PLS)** is a quick, efficient and optimal regression method based on covariance. It is recommended in cases of regression where the number of explanatory variables is high, and where it is likely that there is multicollinearity among the variables, i.e. that the explanatory variables are correlated.

1.6.2.13 Principal Component Analysis / PARAFAC.

Principal Component Analysis (PCA) and PARAFAC (Parallel Factor Analysis) are techniques used in Exploratory Data Analysis (EDA), but they serve different purposes. Let's briefly discuss each of them:

1. **Principal Component Analysis (PCA)**: PCA is a **dimensionality reduction technique** commonly used in EDA and machine learning to transform high-dimensional data into a lower-dimensional representation. It works by finding new orthogonal axes (principal components) that capture the maximum variance in the data. These components are ordered by the amount of variance they explain, with the first component explaining the most variance.

   PCA helps in visualizing data, reducing computational complexity, and eliminating multicollinearity among variables. It is particularly useful when dealing with datasets with many features or to preprocess data before applying machine learning algorithms. By projecting the data onto a smaller set of principal components, you can often preserve most of the relevant information while reducing the dimensionality.

2. **PARAFAC** (Parallel Factor Analysis): PARAFAC, also known as Canonical Polyadic Decomposition (CPD), is a multi-way data analysis method used in EDA and machine learning **for analyzing multi-dimensional datasets**, **such as tensors. Unlike PCA, which operates on matrices (two-dimensional data), PARAFAC can handle higher-order data.**

   In PARAFAC, the goal is to decompose a tensor into a set of factor matrices and a core tensor, which represent the latent structure of the data. **This decomposition can be useful for extracting underlying patterns and relationships from complex datasets with multiple modes (dimensions)**. PARAFAC is commonly used in fields like chemometrics, signal processing, and image analysis.

1.6.2.14 Redundancy Analysis.

Redundancy Analysis is a way to summarize linear relationships in a set of dependent variables that are influenced by a set of independent variables. **It is an extension of multiple linear regression. The method uses a blend of linear regression and Principal Components Analysis(PCA).**

1.7 Another methods used in EDA

Data Type	Univariate Analysis	Bivariate Analysis	Multivariate Analysis
Graphical	Histogram (Numerical)	Scatter Plot (Numerical-Numerical)	3D Scatter Plot (Numerical-Numerical-Numerical)
Graphical	Box Plot (Numerical)	Heatmap (Numerical-Numerical)	3D Surface Plot (Numerical-Numerical-Numerical)
Graphical	Bar Chart (Categorical)	Stacked Bar Chart (Categorical-Categorical)	Mosaic Plot (Categorical-Categorical-Categorical)
Graphical	Pie Chart (Categorical)	Grouped Bar Chart (Categorical-Categorical)	Parallel Coordinates Plot (Numerical-Numerical-Numerical)
Graphical	Line Plot (Numerical)	Grouped Box Plot (Numerical-Categorical)	Radial Plot (Star Plot) (Numerical-Numerical-Numerical)
Graphical	Area Chart (Numerical)	Pair Plot (Scatter Matrix) (Numerical-Numerical)	Andrews Plot (Numerical-Numerical)
Graphical	Density Plot (Numerical)	Joint Plot (Numerical-Numerical)	Correlation Matrix (Numerical-Numerical)
Graphical	Dot Plot (Numerical)	Hexbin Plot (Numerical-Numerical)	3D Parallel Coordinates (Numerical-Numerical-Numerical)
Graphical	Rug Plot (Numerical)	Violin Plot (Numerical-Categorical)	Radar Chart (Categorical-Numerical-Numerical)

Graphical		Grouped Box Plot (Categorical-Numerical)	
Graphical		Boxen Plot (Numerical-Categorical)	
Graphical		Clustered Bar Chart (Categorical-Categorical)	
Graphical		Dendrogram (Categorical-Categorical)	
Graphical		Categorical Scatter Plot (Categorical-Categorical)	
Graphical		Bubble Chart (Numerical-Numerical)	
Non-Graphical	Frequency Table (Categorical)	Cross-Tabulation (Categorical-Categorical)	Three-Way Contingency Table (Categorical-Categorical-Categorical)
Non-Graphical	Mode (Categorical)	Covariance (Numerical-Numerical)	LDA (Linear Discriminant Analysis) (Categorical-Numerical)
Non-Graphical	Summary Statistics (Numerical)	Grouped Summary Statistics (Categorical-Numerical)	Factor Analysis (Numerical-Numerical)
Non-Graphical	Skewness & Kurtosis (Numerical)	ANOVA (Analysis of Variance) (Categorical-Numerical)	PCA (Principal Component Analysis) (Numerical-Numerical)
Non-Graphical	Frequency Distribution (Numerical)	T-Test (Categorical-Numerical)	ICA (Independent Component Analysis) (Numerical-Numerical)
Non-Graphical	Statistical Tests (Shapiro-Wilk	Kolmogorov-Smirnov)	Chi-Square Test (Categorical-Categorical)

Non-Graphical	Value Counts (Categorical)	Fisher's Exact Test (Categorical-Categorical)	Decision Trees (Numerical-Categorical)
Non-Graphical	Percentage Distribution (Categorical)	McNemar's Test (Categorical-Categorical)	Random Forests (Numerical-Categorical)
Non-Graphical		Cramer's V (Categorical-Categorical)	Gradient Boosting (Numerical-Categorical)
Non-Graphical			K-Means Clustering (Numerical)
Non-Graphical			Hierarchical Clustering (Numerical)
Non-Graphical			DBSCAN Clustering (Numerical)
Non-Graphical			t-Distributed Stochastic Neighbor Embedding (t-SNE) (Numerical)
Non-Graphical			UMAP (Uniform Manifold Approximation and Projection) (Numerical)
Non-Graphical			Latent Dirichlet Allocation (LDA) (Text Data)
Non-Graphical			K-Nearest Neighbors (KNN) (Numerical)
Non-Graphical			Apriori Algorithm (Market Basket Analysis) (Categorical)
Non-Graphical			Hierarchical Topic Modeling (Text Data)

1.8 Exploratory Data Analysis Tools

Data specialists perform exploratory data analysis using popular scripting languages for statistics, such as **Python and R**. For effective EDA, data professionals also use a variety of BI (Business Intelligence) tools, including **Qlik Sense, IBM Cognos, and Tableau**.

In programming, we can accomplish EDA using Python, R, SAS. Some of the important packages in Python are:

- Pandas
- NumPy
- Matplotlib
- Seaborn
- Bokeh

Python and R programming languages enable analysts to analyze data better and manipulate it using libraries and packages.

BI tools, incorporating interactive dashboards, robust security, and advanced visualization features, provide data processors with a comprehensive view of data that helps them develop Machine Learning (ML) models

The below visual summarizes 8 powerful EDA tools. These tools automate many redundant steps of EDA and help you profile your data in quick time.

1) SweetViz
- Creates a variety of data visualizations.
- Covers information about missing values, data statistics, etc.
- Integrates with Jupyter Notebook.

2) Pandas-profiling
- Covers info about missing values, data statistics, correlation, etc.
- Produces data alerts.
- Plots data feature interactions.
- Integrates with Jupyter Notebook.

3) DataPrep
- Produces interactive visualizations.
- Typically faster than other common tools.
- Supports Pandas and Dask DataFrames.
- Covers info about missing values, data statistics, correlation, etc.
- Plots data feature interactions.

4) AutoViz
- Interactive Bokeh charts.
- Covers info about missing values, data statistics, correlation, etc.
- Presents data cleaning suggestions.

5) D-Tale
- Allows you to run many common Pandas operations with no code.

- Exports code of analysis.
- Integrates with Jupyter Notebook.
- Covers info about missing values, data statistics, correlation, etc.
- Highlights duplicates, outliers, etc.

6) dabl
- Primarily provides visualizations.
- Covers a wide range of plots:
-- Target distribution
-- Scatter pair plots
-- Histograms, and more.

7) QuickDA
- Get an overview report of the dataset.
- Covers info about missing values, data statistics, correlation, etc.
- Produces data alerts.
- Plots data feature interactions.

8) Lux
- Integrates with Jupyter Notebook.
- Provides visualization recommendations.
- Supports EDA on a subset of columns.

1.9 Some advantages of exploratory data analysis

- Gain Insights into Underlying Trends and Patterns
- Improved Understanding of Variables
- Better Preprocess Data to Save Time
- Make Data-driven Decisions

2 DATA MANIPULATION

Data manipulation is a subset of data processing that includes: selecting, editing, cleaning, and transforming data. In simple words, Data manipulation is the process of modifying data to make it easier to read or better structured.

Generally, the data manipulation method involves **Data manipulation language (DML)**. DML is a programming language that enables the reorganization of data within database software.

There are many data manipulation languages that you can use to do your work more efficiently. We have to be comfortable with them, from basic Python to advanced R, to build accurate models. Furthermore, **Structured Query Language (SQL**) is a popular data manipulation language used to retrieve and manipulate data in a relational database. It aids in communicating with the database.

2.1 Why is data manipulation important? — Objectives

It allows us to access the information which is important to our specific business and goals. This technique can be modified to identify distinct data sets as our business expands or adapts to market conditions. Data manipulation is also useful for discovering and correcting reporting data redundancy.

Thus, data manipulation provides countless benefits to an organization, such as:

Data consistency: Data manipulation allows you to organize your data in a consistent, structured fashion, making it easier to read and understand. With data manipulation and instructions, you can ensure that the data is consistently formatted and saved, even if you do not have a uniform view of the collected data

Data projections: The majority of Business Intelligence (BI) is driven by data, which provides the insights necessary to make informed decisions. Data manipulation makes it easier to create projections, particularly in the financial sector, where future concerns depend on past performance

Delete or neglect data: Data manipulations also help businesses maintain data and unusable data available in the database.

Data interpretation: Complex data offers various challenges, and with data manipulation, it is possible to format that data accordingly. Once it is structured, it can be further transformed into a visual format to better understand the person.

2.2 Data manipulation methods

2.2.1 Data Loading:

Load the raw data into the data analysis environment.

2.2.2 Data Inspection:

Explore the data to understand its structure, quality, and characteristics.

2.2.3 Data Cleaning:

Handle missing values, duplicates, outliers, and data quality issues.

2.2.3.1 Data Cleaning Functions:

There are several functions including

2.2.3.1.1 dropna

The **dropna()** function is a method used to remove missing values from a data structure, typically from a pandas DataFrame or Series in Python. It is a powerful tool for data cleaning and preprocessing, as it allows you to eliminate rows or columns containing missing or NaN (Not a Number) values.

2.2.3.1.2 fillna

The **fillna()** function is a method used to fill missing or NaN (Not a Number) values in a pandas DataFrame or Series in Python. It is commonly used for data preprocessing and imputation, where missing values are replaced with specified values or using various filling techniques.

2.2.3.1.3 drop_duplicates

The drop_duplicates() function is a method in pandas used to remove duplicate rows from a DataFrame. It is particularly useful when dealing with datasets that may contain duplicate entries and you want to keep only the unique rows.

2.2.3.1.4 replace

The replace() function is a method in pandas used to replace values in a DataFrame or Series. It allows you to replace specific values with new values, which is useful for data cleaning, data transformation, and handling missing or incorrect data.

2.2.3.1.5 interpolate

The interpolate() function is a method in pandas used to fill missing values in a DataFrame or Series by using various interpolation techniques. Interpolation is the

process of estimating the missing values based on the available data points, which can be particularly useful for time series data or datasets with missing values.

2.2.3.2 Advanced data imputation techniques

2.2.3.2.1 KNN Imputer

This imputer **utilizes the k-Nearest Neighbors method to replace the missing values in the datasets with the mean value from the parameter 'n_neighbors' nearest neighbors found in the training set**. By default, it uses a Euclidean distance metric to impute the missing values. One thing to note here is that the **KNN Imputer does not recognize text data values. It will generate errors if we do not change these values to numerical values.**

Another critical point here is that the KNN Imptuer is a distance-based imputation method and it requires us to normalize our data. Otherwise, the different scales of our data will lead the KNN Imputer to generate biased replacements for the missing values.

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
df = pd.DataFrame(imputer.fit_transform(df),columns = df.columns)
df.isnull().sum()
```

```
Survived      0
Pclass        0
Age           0
SibSp         0
Parch         0
Fare          0
Sex_male      0
Embarked_Q    0
Embarked_S    0
dtype: int64
```

2.2.3.2.2 Multiple Imputation

The Mean, median, mode imputation, regression imputation, stochastic regression imputation, KNN imputer are all methods that create a single replacement value for each missing entry. **Multiple Imputation (MI)**, rather than a different method, is more like a general approach/framework of **doing the imputation procedure multiple times to create different plausible imputed datasets**. **The key motivation to use MI is that a single imputation cannot reflect sampling variability from both sample data and missing values.**

2.2.3.2.3 Interpolation Methods

Here we take the average of the previous and next value to fill the NULL value

```
df.loc[16:20,]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
16	17	0	3	Rice, Master. Eugene	male	2.0	4	1	382652	29.125	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.000	NaN	S
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female	31.0	1	0	345763	18.000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.225	NaN	C
20	21	0	2	Fynney, Mr. Joseph J	male	35.0	0	0	239865	26.000	NaN	S

```
df.loc[16:20].interpolate()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
16	17	0	3	Rice, Master. Eugene	male	2.0	4	1	382652	29.125	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	16.5	0	0	244373	13.000	NaN	S
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female	31.0	1	0	345763	18.000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	33.0	0	0	2649	7.225	NaN	C
20	21	0	2	Fynney, Mr. Joseph J	male	35.0	0	0	239865	26.000	NaN	S

2.2.3.2.4 Regression Imputation

Regression imputation is a technique for imputing missing values by **predicting them with regression models.** This technique involves using the observed values of the variable with missing data as the dependent variable and the other variables as the independent variables to fit a regression model. The resulting regression model is then used to predict the missing values.

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge

# fit regression model using Bayesian Ridge
imputer = IterativeImputer(estimator=BayesianRidge())

# impute missing values
imputed_data = imputer.fit_transform(data[predictor_vars + missing_vars])

# substitute imputed values for missing values
data[missing_vars] = imputed_data[:, -len(missing_vars):]
```

Regression imputation is classified into **two different versions**: deterministic and stochastic regression imputation.

- **Deterministic regression imputation, replaces missing values with the exact prediction of the regression model.** Random variation (i.e. an error term) around the regression slope is not considered. Imputed values are therefore often too precise and lead to an overestimation of the correlation between X and Y.
- **Stochastic regression imputation** was developed in order to solve this issue of deterministic regression imputation. Stochastic regression imputation **adds a random error term to the predicted value and is therefore able to reproduce the correlation of X and Y more appropriately.**

2.2.3.2.5 Matrix Completion

Matrix Completion is a method for recovering lost information. It originates from machine learning and usually deals with highly sparse matrices. **Missing or unknown data is estimated using the low-rank matrix of the known data.**

Various matrix completion algorithms have been proposed. These includes convex relaxation-based algorithm, gradient-based algorithm, and alternating minimization-based algorithm.

Check this example for a better understanding

Building a Recommendation System using Matrix Completion

2.2.3.2.6 Expectation-Maximization (EM) Algorithm

The expectation-maximization algorithm is an **approach for performing maximum likelihood estimation in the presence of latent variables**. It does this by first estimating the values for the latent variables, then optimizing the model, then repeating these two steps until convergence

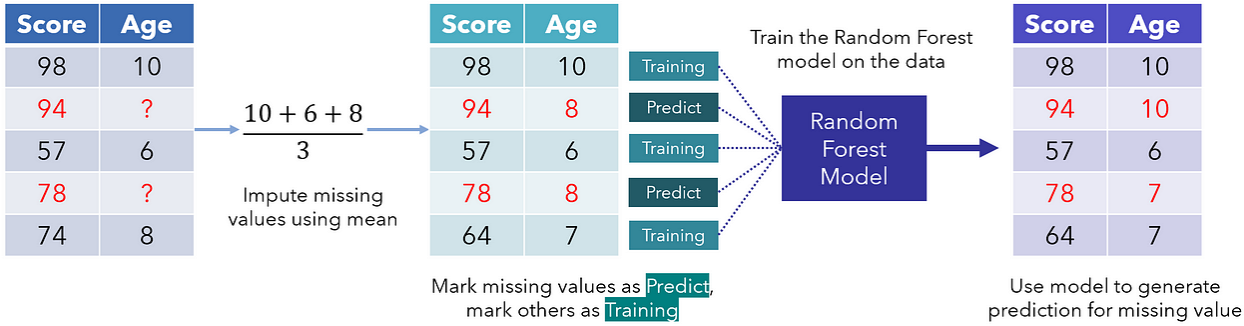
The essence of Expectation-Maximization algorithm is to use the available observed data of the dataset to estimate the missing data and then using that data to update the values of the parameters

2.2.3.2.7 MissForest

MissForest is a machine learning-based data imputation algorithm that operates on the **Random Forest algorithm.**

First, the missing values are filled in using median/mode imputation. Then, we mark the missing values as ‘Predict’ and the others as training rows, which are fed into a Random

Forest model trained to predict, in this case, Age based on Score. The generated prediction for that row is then filled in to produce a transformed dataset.



Assume that the dataset is truncated.

This process of looping through missing data points repeats several times, each iteration improving on better and better data. It's like standing on a pile of rocks while continually adding more to raise yourself: the model uses its current position to elevate itself further.

2.2.4 Data Augmentation

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points.

Note: the augmentation techniques are not limited to images. You can augment audio, video, text, and other types of data too.

When Should You Use Data Augmentation?

1. To prevent models from overfitting.
2. The initial training set is too small.
3. To improve the model accuracy.
4. To Reduce the operational cost of labeling and cleaning the raw dataset.

Limitations of Data Augmentation

- The biases in the original dataset persist in the augmented data.
- Quality assurance for data augmentation is expensive.
- Research and development are required to build a system with advanced applications. For example, generating high-resolution images using GANs can be challenging.
- Finding an effective data augmentation approach can be challenging.

2.2.5 Data Filtering and Subsetting:

Select specific rows or columns based on certain criteria.

2.2.5.1 Data filtering:

Filtering refers to the process of selecting specific rows from a dataset based on certain conditions or criteria. It involves keeping only the rows that meet the specified conditions and discarding the rest. The primary goal of filtering is to reduce the dataset to a subset that is relevant for a particular analysis or task.

```
: df[df.Age<10]
```

:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.00	3	1	349909	21.0750	NaN	S
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.00	1	1	PP 9549	16.7000	G6	S
16	17	0	3	Rice, Master. Eugene	male	2.00	4	1	382652	29.1250	NaN	Q
24	25	0	3	Palsson, Miss. Torborg Danira	female	8.00	3	1	349909	21.0750	NaN	S
43	44	1	2	Laroche, Miss. Simonne Marie Anne Andree	female	3.00	1	2	SC/Paris 2123	41.5792	NaN	C
...	...	...	...	...	...	...	...	...	...	...	...	...
827	828	1	2	Mallet, Master. Andre	male	1.00	0	2	S.C./PARIS 2079	37.0042	NaN	C
831	832	1	2	Richards, Master. George Sibley	male	0.83	1	1	29106	18.7500	NaN	S
850	851	0	3	Andersson, Master. Sigvard Harald Elias	male	4.00	4	2	347082	31.2750	NaN	S
852	853	0	3	Boulos, Miss. Nourelain	female	9.00	1	1	2678	15.2458	NaN	C
869	870	1	3	Johnson, Master. Harold Theodor	male	4.00	1	1	347742	11.1333	NaN	S

62 rows × 12 columns

Data filtering functions:

2.2.5.1.1 dataframe.filter()

Pandas dataframe.filter() function is used to Subset rows or columns of dataframe according to labels in the specified index. Note that this routine does not filter a dataframe on its contents. The filter is applied to the labels of the index.

- items : List of info axis to restrict to (must not all be present)
- like : Keep info axis where “arg in col == True”
- regex : Keep info axis with re.search(regex, col) == True
- ***axis :*** *The axis to filter on. By default this is the info axis, ‘index’ for Series, ‘columns’ for DataFrame*

```
In [31]: # df.filter(items=['PassengerId','Pclass','Parch'])
         # df.filter(regex='^P')
         df.filter(like='P')
```

Out[31]:

	PassengerId	Pclass	Parch
0	1	3	0
1	2	1	0
2	3	3	0
3	4	1	0
4	5	3	0
...	...	...	...
886	887	2	0
887	888	1	0
888	889	3	2
889	890	1	0
890	891	3	0

891 rows × 3 columns

2.2.5.1.2 df.where():

Pandas where() method is used to check a data frame for one or more condition and return the result accordingly. By default, The rows not satisfying the condition are filled with NaN value.

```
df.where(cond=(df.Age<40) & (df.Survived==1) ,other=np.NAN)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2.0	1.0	1.0	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1.0	0.0	PC 17599	71.2833	C85	C
2	3.0	1.0	3.0	Heikkinen, Miss. Laina	female	26.0	0.0	0.0	STON/O2. 3101282	7.9250	NaN	S
3	4.0	1.0	1.0	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1.0	0.0	113803	53.1000	C123	S
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
886	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
887	888.0	1.0	1.0	Graham, Miss. Margaret Edith	female	19.0	0.0	0.0	112053	30.0000	B42	S
888	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
889	890.0	1.0	1.0	Behr, Mr. Karl Howell	male	26.0	0.0	0.0	111369	30.0000	C148	C
890	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

891 rows × 12 columns

2.2.5.1.3 mask():

The mask() method in pandas is used to replace values in a DataFrame based on a condition. It is essentially the opposite of the where() method, which replaces values where the condition is False. The mask() method replaces values where the condition is True.

```
df.mask(cond=(df.Age<40) & (df.Survived==1) ,other=np.NAN)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1.0	0.0	3.0	Braund, Mr. Owen Harris	male	22.0	1.0	0.0	A/5 21171	7.25	NaN	S
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	5.0	0.0	3.0	Allen, Mr. William Henry	male	35.0	0.0	0.0	373450	8.05	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887.0	0.0	2.0	Montvila, Rev. Juozas	male	27.0	0.0	0.0	211536	13.00	NaN	S
887	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
888	889.0	0.0	3.0	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1.0	2.0	W./C. 6607	23.45	NaN	S
889	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
890	891.0	0.0	3.0	Dooley, Mr. Patrick	male	32.0	0.0	0.0	370376	7.75	NaN	Q

891 rows × 12 columns

2.2.5.1.4 4.query():

The query() method in pandas allows you to filter data in a DataFrame using a string expression representing the filtering condition. It provides a more concise and expressive way to perform filtering, especially for complex conditions involving multiple columns.

```
df.query('Age>50 and Sex == "male" and Survived == 1')
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
449	450	1	1	Peuchen, Major. Arthur Godfrey	male	52.0	0	0	113786	30.50	C104	S
570	571	1	2	Harris, Mr. George	male	62.0	0	0	S.W./PP 752	10.50	NaN	S
587	588	1	1	Frolicher-Stehli, Mr. Maxmillian	male	60.0	1	1	13567	79.20	B41	C
630	631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80.0	0	0	27042	30.00	A23	S
647	648	1	1	Simonius-Blumer, Col. Oberst Alfons	male	56.0	0	0	13213	35.50	A26	C
857	858	1	1	Daly, Mr. Peter Denis	male	51.0	0	0	113055	26.55	E17	S

2.2.5.2 Subsetting:

Subsetting refers to the process of selecting specific columns from a dataset. It involves keeping only the columns that are of interest for a particular analysis or task and discarding the rest. Subsetting is useful when you want to focus on a particular set of attributes or features in your data.

```
In [25]: df[['Name','Age']]
```

Out[25]:

	Name	Age
0	Braund, Mr. Owen Harris	22.0
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0
2	Heikkinen, Miss. Laina	26.0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0
4	Allen, Mr. William Henry	35.0
...	...	...
886	Montvila, Rev. Juozas	27.0
887	Graham, Miss. Margaret Edith	19.0
888	Johnston, Miss. Catherine Helen "Carrie"	NaN
889	Behr, Mr. Karl Howell	26.0
890	Dooley, Mr. Patrick	32.0

891 rows × 2 columns

Subsetting functions:

2.2.5.2.1 loc and iloc:

DataFrame.loc:

Label-based indexing used for subsetting rows and columns by their labels or index names.

```
df.loc[1:10,'Survived':'Age']
```

	Survived	Pclass	Name	Sex	Age
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	1	3	Heikkinen, Miss. Laina	female	26.0
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
4	0	3	Allen, Mr. William Henry	male	35.0
5	0	3	Moran, Mr. James	male	NaN
6	0	1	McCarthy, Mr. Timothy J	male	54.0
7	0	3	Palsson, Master. Gosta Leonard	male	2.0
8	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0
9	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0
10	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0

DataFrame.iloc:

Integer-based indexing used for subsetting rows and columns by their integer positions (zero-based index).

```
df.iloc[1:11,1:6]
```

	Survived	Pclass	Name	Sex	Age
1	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0
2	1	3	Heikkinen, Miss. Laina	female	26.0
3	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0
4	0	3	Allen, Mr. William Henry	male	35.0
5	0	3	Moran, Mr. James	male	NaN
6	0	1	McCarthy, Mr. Timothy J	male	54.0
7	0	3	Palsson, Master. Gosta Leonard	male	2.0
8	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0
9	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0
10	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0

2.2.5.2.2 at and iat:

DataFrame.at:

Access a single value for a row/column label pair.

Similar to loc, in that both provide label-based lookups. Use at if you only need to get or set a single value in a DataFrame or Series.

DataFrame.iat:

Access a single value for a row/column pair by integer position.

Similar to iloc, in that both provide integer-based lookups. Use iat if you only need to get or set a single value in a DataFrame or Series.

```
In [7]: df.at[1,'Name']
Out[7]: 'Cumings, Mrs. John Bradley (Florence Briggs Thayer)'

In [ ]:

In [9]: df.iat[1,3]
Out[9]: 'Cumings, Mrs. John Bradley (Florence Briggs Thayer)'
```

2.2.5.2.3 3.DataFrame.xs

Return cross-section from the Series/DataFrame.

This method takes a key argument to select data at a particular level of a MultiIndex.

```
In [13]: df
```

Out[13]:

		Value
Group	Letter	
A	X	10
	Y	20
B	X	30
	Y	40
C	X	50
	Y	60

```
In [14]: df.xs('A')
```

Out[14]:

	Value
Letter	
X	10
Y	20

2.2.6 Data Sorting:

Data sorting in the context of data manipulation refers to the process of arranging the data in a specific order based on one or more columns in a DataFrame. Sorting is essential for data analysis as it helps in visualizing trends, identifying patterns, and making the data more presentable.

Dataframe sorting functions:

2.2.6.1 sort_values():

In pandas, you can use the sort_values() method to perform data sorting based on one or more columns. The sort_values() method allows you to specify the column(s) you want to sort by and the sorting order (ascending or descending).

```
df.sort_values(by='Age',ascending=True)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
803	804	1	3	Thomas, Master. Assad Alexander	male	0.42	0	1	2625	8.5167	NaN	C
755	756	1	2	Hamalainen, Master. Viljo	male	0.67	1	1	250649	14.5000	NaN	S
644	645	1	3	Baclini, Miss. Eugenie	female	0.75	2	1	2666	19.2583	NaN	C
469	470	1	3	Baclini, Miss. Helene Barbara	female	0.75	2	1	2666	19.2583	NaN	C
78	79	1	2	Caldwell, Master. Alden Gates	male	0.83	0	2	248738	29.0000	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
859	860	0	3	Razi, Mr. Raihed	male	NaN	0	0	2629	7.2292	NaN	C
863	864	0	3	Sage, Miss. Dorothy Edith "Dolly"	female	NaN	8	2	CA. 2343	69.5500	NaN	S
868	869	0	3	van Melkebeke, Mr. Philemon	male	NaN	0	0	345777	9.5000	NaN	S
878	879	0	3	Laleff, Mr. Kristo	male	NaN	0	0	349217	7.8958	NaN	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S

891 rows × 12 columns

2.2.6.2 sort_index():

To sort a DataFrame by its index, you can use the sort_index() method in pandas. This method sorts the rows of the DataFrame based on the values in the index. It is particularly useful when the index represents dates, times, or any other ordered values.

```
d = df.loc[[4,2,3,0,1]]
d
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

```
d.sort_index(ascending=False)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S

2.2.7 Data Aggregation and Grouping:

Group data based on one or more variables and calculate summary statistics

Data aggregation

Data aggregation is a process in data analysis where raw data is combined and summarized to provide meaningful insights and reduce the data to a more manageable form. It involves grouping data based on one or more variables and then applying one or more aggregation functions to calculate summary statistics for each group. Data aggregation is a crucial step in understanding the patterns, trends, and characteristics of a dataset.

Here are the key steps involved in data aggregation:

2.2.7.1 Grouping:

Data is grouped based on specific columns or variables that have common characteristics. For example, grouping data based on categories, dates, or any other relevant attributes. Once the data is grouped, you can perform various operations within each group, such as aggregations, transformations, or filtering. Grouping is essential for understanding the relationships and patterns in your data, especially when dealing with large and complex datasets.

2.2.7.1.1 DataFrame. groupby():

In Python, the pandas library provides the groupby() function, which is a powerful tool for grouping data in a DataFrame based on specific columns. The groupby() function allows you to create a "GroupBy" object that represents the grouped data. You can then apply various operations to this GroupBy object to analyze the data within each group.

2.2.7.2 Aggregation Functions:

After grouping the data, one or more aggregation functions are applied to calculate summary statistics for each group. Common aggregation functions include mean, sum, median, count, min, max, standard deviation, and more.

agg():

The agg() function in pandas is a powerful method used for data aggregation. It allows you to perform multiple aggregation operations on a DataFrame or GroupBy object simultaneously.

common aggregation functions

2.2.7.2.1 count:

Calculates the number of non-missing (non-null) values in the specified column(s) or DataFrame. It returns the count of non-null elements within each group when used with groupby().

2.2.7.2.2 sum:

Calculates the sum of the numeric values in the specified column(s) or DataFrame. It returns the sum of elements within each group when used with groupby().

2.2.7.2.3 mean:

Calculates the arithmetic mean (average) of the numeric values in the specified column(s) or DataFrame. It returns the mean value within each group when used with groupby().

2.2.7.2.4 median:

Calculates the median value (the middle value when data is sorted) of the numeric values in the specified column(s) or DataFrame. It returns the median value within each group when used with groupby().

2.2.7.2.5 min:

Calculates the minimum value of the numeric values in the specified column(s) or DataFrame. It returns the minimum value within each group when used with groupby().

2.2.7.2.6 max:

Calculates the maximum value of the numeric values in the specified column(s) or DataFrame. It returns the maximum value within each group when used with groupby().

2.2.7.2.7 std:

Calculates the standard deviation of the numeric values in the specified column(s) or DataFrame. It measures the dispersion or spread of data from the mean. It returns the standard deviation within each group when used with groupby()

2.2.7.2.8 var:

Calculates the variance of numeric values in the specified column(s) or DataFrame. It returns the variance within each group when used with groupby()

```
# group dataframe based on gender and perform some data aggegation functions
df.groupby('Sex')['Age'].agg(['count','min','median','mean','max','var','std','sum',])
```

Sex	count	min	median	mean	max	var	std	sum
female	261	0.75	27.0	27.915709	63.0	199.096233	14.110146	7286.00
male	453	0.42	29.0	30.726645	80.0	215.449579	14.678201	13919.17

2.2.7.2.9 first():

Returns the first element of the specified column or DataFrame when used with groupby(). It is useful when you want to extract the first occurrence of a value within each group.

```
grouped_data = df.groupby(['Pclass', 'Sex'])
grouped_data.first()
```

Pclass	Sex	PassengerId	Survived	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	female	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	1	0	PC 17599	71.2833	C85	C
	male	7	0	McCarthy, Mr. Timothy J	54.0	0	0	17463	51.8625	E46	S
2	female	10	1	Nasser, Mrs. Nicholas (Adele Achem)	14.0	1	0	237736	30.0708	F33	C
	male	18	1	Williams, Mr. Charles Eugene	35.0	0	0	244373	13.0000	D56	S
3	female	3	1	Heikkinen, Miss. Laina	26.0	0	0	STON/O2. 3101282	7.9250	G6	S
	male	1	0	Braund, Mr. Owen Harris	22.0	1	0	A/5 21171	7.2500	F G73	S

2.2.7.2.10 last():

last: Returns the last element of the specified column or DataFrame when used with groupby(). It is useful when you want to extract the last occurrence of a value within each group.

```
grouped_data.last()
```

Pclass	Sex	PassengerId	Survived	Name	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	female	888	1	Graham, Miss. Margaret Edith	19.0	0	0	112053	30.00	B42	S
	male	890	1	Behr, Mr. Karl Howell	26.0	0	0	111369	30.00	C148	C
2	female	881	1	Shelley, Mrs. William (Imanita Parrish Hall)	25.0	0	1	230433	26.00	E77	S
	male	887	0	Montvila, Rev. Juozas	27.0	0	0	211536	13.00	F2	S
3	female	889	0	Johnston, Miss. Catherine Helen "Carrie"	39.0	1	2	W./C. 6607	23.45	E121	S
	male	891	0	Dooley, Mr. Patrick	32.0	0	0	370376	7.75	F38	Q

2.2.7.2.11 nunique:

nunique: Calculates the number of unique (distinct) values in the specified column or DataFrame when used with groupby(). It returns the count of unique elements within each group.

```
# Number of unique values in 'Age' within each group
grouped_data['Age'].nunique()
```

```
Pclass  Sex
1       female    41
        male      49
2       female    40
        male      44
3       female    46
        male      63
Name: Age, dtype: int64
```

2.2.7.2.12 cumsum:

Calculates the cumulative sum of numeric values in the specified column or DataFrame. It returns a new Series with the cumulative sum at each row.

2.2.7.2.13 cummin:

Calculates the cumulative minimum of numeric values in the specified column or DataFrame. It returns a new Series with the cumulative minimum at each row.

2.2.7.2.14 cummax:

Calculates the cumulative maximum of numeric values in the specified column or DataFrame. It returns a new Series with the cumulative maximum at each row.

2.2.7.2.15 cumprod:

Calculates the cumulative product of numeric values in the specified column or DataFrame. It returns a new Series with the cumulative product at each row.

2.2.7.3 Result Summary:

The results of the aggregation are typically presented in a summarized format, such as a table or a new DataFrame, where each row represents a group, and each column represents a summary statistic.

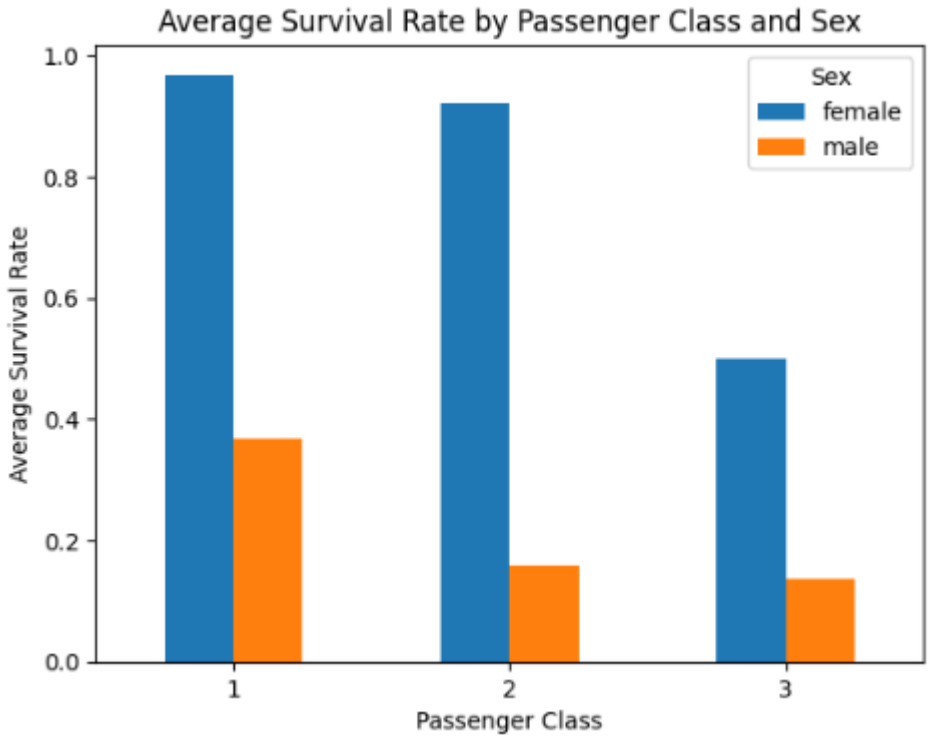
```
grouped_data = df.groupby(['Pclass', 'Sex'])
summary_stats = grouped_data.agg({
    'Age': ['min','mean', 'median', 'max'],
    'Fare': ['min','mean', 'median', 'max']
})
summary_stats
```

		Age				Fare			
		min	mean	median	max	min	mean	median	max
Pclass	Sex								
1	female	2.00	34.611765	35.0	63.0	25.9292	106.125798	82.66455	512.3292
	male	0.92	41.281386	40.0	80.0	0.0000	67.226127	41.26250	512.3292
2	female	2.00	28.722973	28.0	57.0	10.5000	21.970121	22.00000	65.0000
	male	0.67	30.740707	30.0	70.0	0.0000	19.741782	13.00000	73.5000
3	female	0.75	21.750000	21.5	63.0	6.7500	16.118810	12.47500	69.5500
	male	0.42	26.507589	25.0	74.0	0.0000	12.661633	7.92500	69.5500

2.2.7.4 Visual Analysis:

The summarized data can be used for data visualization, further analysis, or making informed decisions based on the insights obtained from the aggregation.

```
:
import pandas as pd
import matplotlib.pyplot as plt

# Grouping by 'Pclass' and 'Sex'
grouped_data = df.groupby(['Pclass', 'Sex'])

# Calculating the mean survival rate within each group
mean_survival_rate_by_group = grouped_data['Survived'].mean()

# Data Visualization: Bar plot to compare mean survival rates among different passenger classes and genders
mean_survival_rate_by_group.unstack().plot(kind='bar', rot=0)
plt.title('Average Survival Rate by Passenger Class and Sex')
plt.xlabel('Passenger Class')
plt.ylabel('Mean Survival Rate')
plt.show()
```



2.2.8 Data Transformation:

Data Transformation Functions:

2.2.8.1 Mathematical functions (e.g., add, sub, mul, div)

2.2.8.1.1 add()

The **pandas addition function** performs the addition of dataframes. The addition is performed element-wise.

```
In [70]: df = pd.DataFrame({'Experience': [3,2,3,2,4,2,2,5,3,2],
                            'Salary': [15000, 16000, 26500, 30654, 20982, 40000, 50000, 32000, 20200, 52012],
                            })
         df
```

Out[70]:

	Experience	Salary
0	3	15000
1	2	16000
2	3	26500
3	2	30654
4	4	20982
5	2	40000
6	2	50000
7	5	32000
8	3	20200
9	2	52012

```
In [71]: df1 = pd.DataFrame({'Experience': [1,1,1,1,1,1,1,1,1,1],
                            'Salary': [1500, 1600, 2650, 3065, 2098, 4000, 5000, 3200, 2000, 2027]
                            })
         df1
```

Out[71]:

	Experience	Salary
0	1	1500
1	1	1600
2	1	2650
3	1	3065
4	1	2098
5	1	4000
6	1	5000
7	1	3200
8	1	2000
9	1	2027

```
In [72]: df.add(df1)
```

Out[72]:

	Experience	Salary
0	4	16500
1	3	17600
2	4	29150
3	3	33719
4	5	23080
5	3	44000
6	3	55000
7	6	35200
8	4	22200
9	3	54039

2.2.8.1.2 sub()

The subtract function of pandas is used to perform subtract operation on dataframes.

```
In [73]: df.sub(df1)
```

Out[73]:

	Experience	Salary
0	2	13500
1	1	14400
2	2	23850
3	1	27589
4	3	18884
5	1	36000
6	1	45000
7	4	28800
8	2	18200
9	1	49985

2.2.8.1.3 mul()

The multiplication function of pandas is used to perform multiplication operations on dataframes.

```
In [84]: df2 = pd.DataFrame({'speed': [50, 75, 100]},
                            index=['Audi', 'Jaguar', 'BMW'])
         df2
```

Out[84]:

	speed
Audi	50
Jaguar	75
BMW	100

```
In [102]: df21 = pd.DataFrame({'speed': [5, 5, 5]},
                              index=['Audi', 'Jaguar', 'BMW'])
          df21
```

Out[102]:

	speed
Audi	5
Jaguar	5
BMW	5

```
In [103]: df2.mul(df21)
```

Out[103]:

	speed
Audi	250
Jaguar	375
BMW	500

2.2.8.1.4 div()

The division function of pandas is used to perform division operation on dataframes.

```
In [88]: df2.div(5)
```

Out[88]:

	speed
Audi	10.0
Jaguar	15.0
BMW	20.0

```
In [99]: df3 = pd.DataFrame({'speed': [50, 75, 100],
                             'weight': [250, 200, 150],
                            'price':[300000,400000,450000],
                            'limit':[2,2,2]},

                            index=['Audi', 'Jaguar', 'BMW'])
         df3
```

Out[99]:

	speed	weight	price	limit
Audi	50	250	300000	2
Jaguar	75	200	400000	2
BMW	100	150	450000	2

```
In [101]: df3['speed'].div(df3['limit'])
```

```
Out[101]: Audi      25.0
          Jaguar    37.5
          BMW       50.0
          dtype: float64
```

2.2.8.2 Apply

Apply allow the users to pass a function and apply it on every single value of the Pandas series.

Out[6]:

	EmployeeName	Department	HireDate	Sex	Birthdate	Weight	Height	Kids
0	Callen Dunkley	Accounting	2010	M	04/09/1982	78	176	2
1	Sarah Rayner	Engineering	2018	F	14/04/1981	80	160	1
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	66	169	0
3	Kaycee Acosta	HR	2014	F	08/01/1986	67	157	1
4	Henri Conroy	HR	2014	M	10/10/1988	90	185	1
5	Emma Peralta	HR	2018	F	12/11/1992	57	164	0
6	Martin Butt	Data Science	2020	M	10/04/1991	115	195	2
7	Alex Jensen	Data Science	2018	M	16/07/1995	87	180	0
8	Kim Howarth	Accounting	2020	M	08/10/1992	95	174	3
9	Jane Burnett	Data Science	2012	F	11/10/1979	57	165	1

```
In [9]: mask = data['HireDate'].apply(lambda x: date.today().year - x >= 10)
        print(mask)
```

```
0     True
1    False
2     True
3    False
4    False
5    False
6    False
7    False
8    False
9     True
Name: HireDate, dtype: bool
```

2.2.8.3 Map:

pandas.map() is used to map values from two series having one column same.

map()

```
In [106]: s = pd.Series(['fox', 'cow', 'horse', 'dog'])
          s.map('It is a {}'.format)

Out[106]: 0      It is a fox
          1      It is a cow
          2    It is a horse
          3      It is a dog
          dtype: object
```

2.2.8.4 Transform:

Transform function returns a self-produced dataframe with transformed values after applying the function specified in its parameter. This dataframe has the same length as the passed dataframe.

Out[141]:

	A	B	C
0	3	20	16
1	3	18	18
2	6	20	18
3	4	18	26
4	2	15	30
5	3	12	22
6	8	18	21
7	3	18	25
8	8	20	19
9	3	18	14

```
In [142]: x.transform(multi)
```

Out[142]:

	A	B	C
0	9	60	48
1	9	54	54
2	18	60	54
3	12	54	78
4	6	45	90
5	9	36	66
6	24	54	63
7	9	54	75
8	24	60	57
9	9	54	42

2.2.8.5 cut:

Pandas cut() function is used to separate the array elements into different bins . The cut function is mainly used to perform statistical analysis on scalar data.

```
In [5]: import pandas as pd
        import numpy as np
        import random
        info_nums = pd.DataFrame({'num': np.random.randint(1, 10, 7)})
        print(info_nums)
        info_nums['nums_labels'] = pd.cut(x=info_nums['num'], bins=[1, 7, 10], labels=['Lows', 'Highs'], right=False)
        print(info_nums)
        print(info_nums['nums_labels'].unique())
```

```
   num
0    7
1    8
2    4
3    7
4    9
5    4
6    8
   num nums_labels
0    7       Highs
1    8       Highs
2    4        Lows
3    7       Highs
4    9       Highs
5    4        Lows
6    8       Highs
['Highs', 'Lows']
Categories (2, object): ['Lows' < 'Highs']
```

2.2.8.6 qcut:

Qcut (quantile-cut) differs from cut in the sense that, in qcut, the number of elements in each bin will be roughly the same, but this will come at the cost of differently sized interval widths. On the other hand, in cut, the bin edges were equal sized (when we specified bins=3) with uneven number of elements in each bin or group.

```
: import pandas as pd
  import numpy as np
  info_nums = pd.DataFrame({'num': np.random.randint(1, 10, 7)})
  print(info_nums)
  info_nums['nums_labels'] = pd.qcut(x=info_nums['num'], q=4, duplicates='drop')
  print(info_nums)
  print(info_nums['nums_labels'].unique())
```

```
   num
0    3
1    4
2    3
3    8
4    6
5    3
6    9
   num   nums_labels
0    3  (2.999, 4.0]
1    4  (2.999, 4.0]
2    3  (2.999, 4.0]
3    8    (7.0, 9.0]
4    6    (4.0, 7.0]
5    3  (2.999, 4.0]
6    9    (7.0, 9.0]
[(2.999, 4.0], (7.0, 9.0], (4.0, 7.0]]
Categories (3, interval[float64, right]): [(2.999, 4.0] < (4.0, 7.0] < (7.0, 9.0]]
```

2.2.8.7 get_dummies:

The get_dummies() function is used to convert categorical variable into dummy/indicator variables. The function iterates over the object that is passed and checks if the element at the particular index matches the column heading. If it does, it encodes it as a 1.Otherwise, it assigns it a 0.

Out[27]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

In [29]:
```
dfx = pd.get_dummies(df['Sex'])
dfx
```

Out[29]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

891 rows × 2 columns

2.2.8.8 other transformations

2.2.8.8.1 1.Log Transformation:

Log and natural logarithmic value of a column in pandas python is carried out using log2(), log10() and log()function of NumPy.

- Get the natural logarithmic value of column in pandas (natural log – loge())
- Get the logarithmic value of the column in pandas with base 2 – log2()
- Get the logarithmic value of the column in pandas with base 10 – log10()

	EmployeeName	Department	HireDate	Sex	Birthdate	Salary
0	Callen Dunkley	Accounting	2010	M	04/09/1982	1500
1	Sarah Rayner	Engineering	2018	F	14/04/1981	1600
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	2650
3	Kaycee Acosta	HR	2014	F	08/01/1986	3065
4	Henri Conroy	HR	2014	M	10/10/1988	2098
5	Emma Peralta	HR	2018	F	12/11/1992	4000
6	Martin Butt	Data Science	2020	M	10/04/1991	5000
7	Alex Jensen	Data Science	2018	M	16/07/1995	3200
8	Kim Howarth	Accounting	2020	M	08/10/1992	2000
9	Jane Burnett	Data Science	2012	F	11/10/1979	2027

```
In [35]: data['log'] = np.log(data['Salary'])
```

```
In [37]: data['log2'] = np.log2(data['Salary'])
```

```
In [39]: data['log10'] = np.log10(data['Salary'])
```

```
In [40]: data
```

Out[40]:

	EmployeeName	Department	HireDate	Sex	Birthdate	Salary	log	log2	log10
0	Callen Dunkley	Accounting	2010	M	04/09/1982	1500	7.313220	10.550747	3.176091
1	Sarah Rayner	Engineering	2018	F	14/04/1981	1600	7.377759	10.643856	3.204120
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	2650	7.882315	11.371777	3.423246
3	Kaycee Acosta	HR	2014	F	08/01/1986	3065	8.027803	11.581671	3.486430
4	Henri Conroy	HR	2014	M	10/10/1988	2098	7.648740	11.034799	3.321805
5	Emma Peralta	HR	2018	F	12/11/1992	4000	8.294050	11.965784	3.602060
6	Martin Butt	Data Science	2020	M	10/04/1991	5000	8.517193	12.287712	3.698970
7	Alex Jensen	Data Science	2018	M	16/07/1995	3200	8.070906	11.643856	3.505150
8	Kim Howarth	Accounting	2020	M	08/10/1992	2000	7.600902	10.965784	3.301030
9	Jane Burnett	Data Science	2012	F	11/10/1979	2027	7.614312	10.985130	3.306854

2.2.8.8.2 Square Root Transformation:

Python numpy.sqrt() function is used to return the non-negative square root of an array element-wise (for each element of the array).

Out[51]:

	EmployeeName	Department	HireDate	Sex	Birthdate	Salary	Hours
0	Callen Dunkley	Accounting	2010	M	04/09/1982	1500	3
1	Sarah Rayner	Engineering	2018	F	14/04/1981	1600	13
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	2650	8
3	Kaycee Acosta	HR	2014	F	08/01/1986	3065	20
4	Henri Conroy	HR	2014	M	10/10/1988	2098	1
5	Emma Peralta	HR	2018	F	12/11/1992	4000	6
6	Martin Butt	Data Science	2020	M	10/04/1991	5000	12
7	Alex Jensen	Data Science	2018	M	16/07/1995	3200	17
8	Kim Howarth	Accounting	2020	M	08/10/1992	2000	4
9	Jane Burnett	Data Science	2012	F	11/10/1979	2027	10

In [53]:
```
data1['Squared'] = np.sqrt(data1['Hours'])
data1
```

Out[53]:

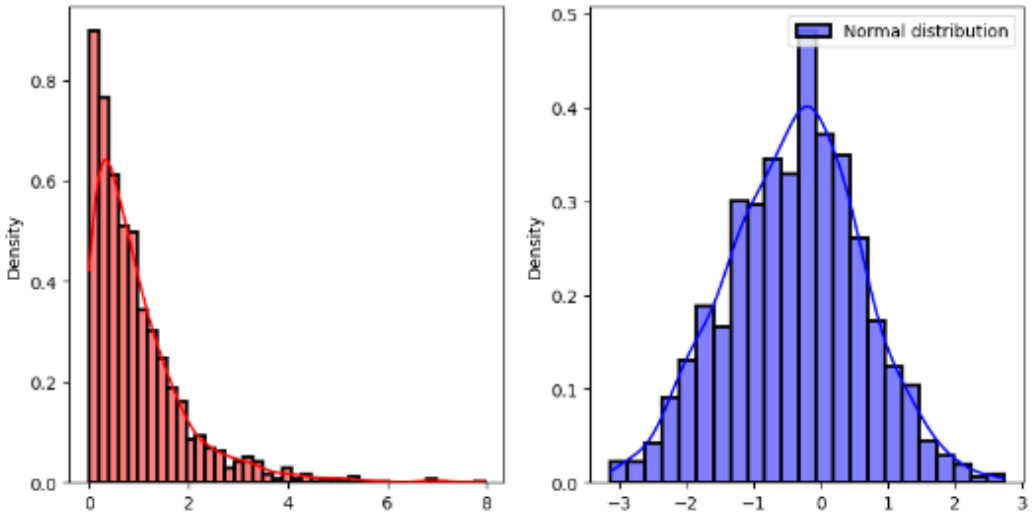
	EmployeeName	Department	HireDate	Sex	Birthdate	Salary	Hours	Squared
0	Callen Dunkley	Accounting	2010	M	04/09/1982	1500	3	1.732051
1	Sarah Rayner	Engineering	2018	F	14/04/1981	1600	13	3.605551
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	2650	8	2.828427
3	Kaycee Acosta	HR	2014	F	08/01/1986	3065	20	4.472136
4	Henri Conroy	HR	2014	M	10/10/1988	2098	1	1.000000
5	Emma Peralta	HR	2018	F	12/11/1992	4000	6	2.449490
6	Martin Butt	Data Science	2020	M	10/04/1991	5000	12	3.464102
7	Alex Jensen	Data Science	2018	M	16/07/1995	3200	17	4.123106
8	Kim Howarth	Accounting	2020	M	08/10/1992	2000	4	2.000000
9	Jane Burnett	Data Science	2012	F	11/10/1979	2027	10	3.162278

2.2.8.8.3 Box-Cox Transformation:

A Box cox transformation is defined as a way to transform non-normal dependent variables in our data to a normal shape through which we can run a lot more tests than we could have.

```
import numpy as np
from scipy import stats

# importing the plotting modules
import seaborn as sns
import matplotlib.pyplot as plt

# generating the random non-normal data (exponential)
originalData = np.random.exponential(size = 1200)

# transforming the data into normal data and getting the lambda value
fittedData, lambdaValue = stats.boxcox(originalData)

# creating the axes to draw plots of datasets
fig, ax = plt.subplots(1, 2)

#plotting the non-normal or original data using histplot() function
sns.histplot(originalData, kde = True,stat="density",linewidth=2,label = "Non-Normal", color ="red", ax = ax[0])

#plotting the normal or original data using the histplot() function
sns.histplot(fittedData, kde = True,stat="density",linewidth=2,label = "Normal distribution", color ="blue", ax = ax[1])

# adding legends to the subplots
plt.legend(loc = "upper right")
# rescaling the subplots
fig.set_figheight(5)
fig.set_figwidth(10)
print(f"Lambda value used for Transformation: {lambdaValue}")
#displaying the plots
plt.show()
```

```
Lambda value used for Transformation: 0.2567422711001565
```



If w is our transformed variable and “y” is our target variable, then the Box-Cox transformation equation looks like this:

$$w_t = \begin{cases} \log(y_t) & \text{if } \lambda = 0; \\ (y_t^{\lambda} - 1)/\lambda & \text{otherwise.} \end{cases}$$

2.2.8.8.4 Min-Max Scaling (Normalization):

MinMaxScaler is a data preprocessing technique used to normalize numerical data within a specified range, typically between 0 and 1. The purpose of MinMaxScaler is to transform each feature (column) of the data independently, so they all lie in the specified range. This scaling is particularly useful when the features have different ranges, and you want to bring them all to a common scale.

Min-Max Scale - MinMaxScaler()

n [417]:
```
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
df1[['Age','Fare']] = mm.fit_transform(df1[['Age','Fare']])
df1
```

ut[417]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	1	0	3	0.271174	1	0	0.014151	0	1	0	0	1
1	2	1	1	0.472229	1	0	0.139136	1	0	1	0	0
2	3	1	3	0.321438	0	0	0.015469	1	0	0	0	1
3	4	1	1	0.434531	1	0	0.103644	1	0	0	0	1
4	5	0	3	0.434531	0	0	0.015713	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	0.334004	0	0	0.025374	0	1	0	0	1
887	888	1	1	0.233476	0	0	0.058556	1	0	0	0	1
888	889	0	3	0.367921	1	2	0.045771	1	0	0	0	1
889	890	1	1	0.321438	0	0	0.058556	0	1	1	0	0
890	891	0	3	0.396833	0	0	0.015127	0	1	0	1	0

891 rows × 12 columns

2.2.8.8.5 5.Z-Score Scaling (Standardization):

The Z-score is a statistical measure that represents the number of standard deviations a data point is from the mean of a dataset. It standardizes data, allowing comparisons between different datasets with different scales and distributions. Positive Z-scores indicate data points above the mean, while negative Z-scores indicate data points below the mean. Z-scores are used to identify outliers, assess relative positions in a dataset, and simplify statistical analysis in various fields.

Out[59]:

	A	B	C
0	6	15	30
1	3	13	19
2	7	17	19
3	5	20	16
4	8	18	28
5	9	13	29
6	4	18	28
7	7	18	29
8	2	18	29
9	6	13	27

In [62]:
```
x['A'] = stats.zscore(x['A'])
x['B'] = stats.zscore(x['B'])
x['C'] = stats.zscore(x['C'])
x
```

Out[62]:

	A	B	C
0	0.142857	-0.530281	0.926696
1	-1.285714	-1.346098	-1.289317
2	0.619048	0.285536	-1.289317
3	-0.333333	1.509261	-1.893684
4	1.095238	0.693444	0.523785
5	1.571429	-1.346098	0.725241
6	-0.809524	0.693444	0.523785
7	0.619048	0.693444	0.725241
8	-1.761905	0.693444	0.725241
9	0.142857	-1.346098	0.322329

2.2.8.8.6 Rank Transformation:

rank() method returns a rank of every respective index of a series passed. The rank is returned on the basis of position after sorting.

	EmployeeName	Department	HireDate	Sex	Birthdate	Salary
0	Callen Dunkley	Accounting	2010	M	04/09/1982	1500
1	Sarah Rayner	Engineering	2018	F	14/04/1981	1600
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	2650
3	Kaycee Acosta	HR	2014	F	08/01/1986	3065
4	Henri Conroy	HR	2014	M	10/10/1988	2098
5	Emma Peralta	HR	2018	F	12/11/1992	4000
6	Martin Butt	Data Science	2020	M	10/04/1991	5000
7	Alex Jensen	Data Science	2018	M	16/07/1995	3200
8	Kim Howarth	Accounting	2020	M	08/10/1992	2000
9	Jane Burnett	Data Science	2012	F	11/10/1979	2027

```
data2['EmployeeRank'] = data2['EmployeeName'].rank()
```

```
data2
```

	EmployeeName	Department	HireDate	Sex	Birthdate	Salary	EmployeeRank
0	Callen Dunkley	Accounting	2010	M	04/09/1982	1500	2.0
1	Sarah Rayner	Engineering	2018	F	14/04/1981	1600	10.0
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	2650	6.0
3	Kaycee Acosta	HR	2014	F	08/01/1986	3065	7.0
4	Henri Conroy	HR	2014	M	10/10/1988	2098	4.0
5	Emma Peralta	HR	2018	F	12/11/1992	4000	3.0
6	Martin Butt	Data Science	2020	M	10/04/1991	5000	9.0
7	Alex Jensen	Data Science	2018	M	16/07/1995	3200	1.0
8	Kim Howarth	Accounting	2020	M	08/10/1992	2000	8.0
9	Jane Burnett	Data Science	2012	F	11/10/1979	2027	5.0

```
data2.sort_values(by= 'EmployeeName', inplace= True)
data2
```

	EmployeeName	Department	HireDate	Sex	Birthdate	Salary	EmployeeRank
7	Alex Jensen	Data Science	2018	M	16/07/1995	3200	1.0
0	Callen Dunkley	Accounting	2010	M	04/09/1982	1500	2.0
5	Emma Peralta	HR	2018	F	12/11/1992	4000	3.0
4	Henri Conroy	HR	2014	M	10/10/1988	2098	4.0
9	Jane Burnett	Data Science	2012	F	11/10/1979	2027	5.0
2	Jeanette Sloan	Engineering	2012	F	06/05/1997	2650	6.0
3	Kaycee Acosta	HR	2014	F	08/01/1986	3065	7.0
8	Kim Howarth	Accounting	2020	M	08/10/1992	2000	8.0
6	Martin Butt	Data Science	2020	M	10/04/1991	5000	9.0
1	Sarah Rayner	Engineering	2018	F	14/04/1981	1600	10.0

2.2.8.8.7 Binarization:

Binarization is a preprocessing technique which is used when we need to convert the data into binary numbers i.e., when we need to binarize the data.

	Values
0	28
1	45
2	18
3	20
4	22
5	31
6	42
7	10
8	28
9	20
10	38
11	40
12	21
13	39

```
: from sklearn.preprocessing import Binarizer
  print(Binarizer(threshold=25).transform(dx))
```

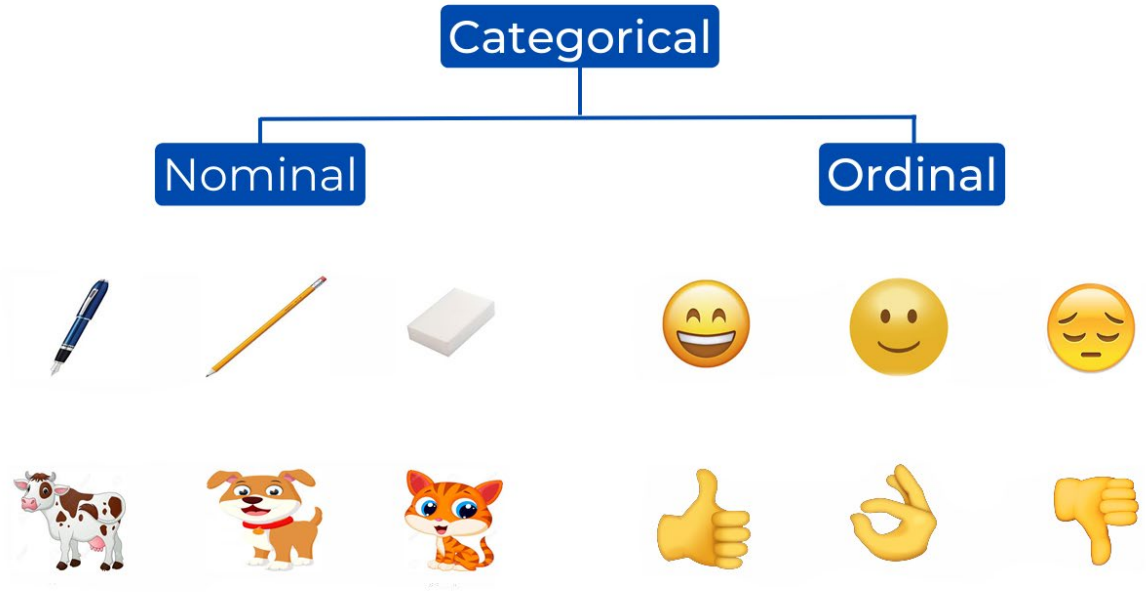
```
[[1]
 [1]
 [0]
 [0]
 [0]
 [1]
 [1]
 [0]
 [1]
 [0]
 [1]
 [1]
 [0]
 [1]]
```

2.2.8.8.8 8.Feature Encoding methods

Machine learning models can only work with numerical values. For this reason, it is necessary to transform the categorical values of the relevant features into numerical ones. This process is called feature encoding.

For this we can use the scikit library or other libraries like category_ecnoders

Categorical values are classified as



- **Ordinal Data:** The categories have an inherent order
- **Nominal Data:** The categories do not have an inherent order

2.2.8.8.8.1 Label Encoding:

Label encoding is a process of converting categorical data or text labels into numerical representations. In this technique, each unique label in the categorical data is assigned a unique integer value.

LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df.iloc[:,3] = le.fit_transform(df.iloc[:,3])
dummies = pd.get_dummies(df[['Sex','Embarked']])
dummies
```

	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	0	1	0	0	1
1	1	0	1	0	0
2	1	0	0	0	1
3	1	0	0	0	1
4	0	1	0	0	1
...	...	...	...	...	...
886	0	1	0	0	1
887	1	0	0	0	1
888	1	0	0	0	1
889	0	1	1	0	0
890	0	1	0	1	0

891 rows × 5 columns

```
df = df.drop(df[['Sex','Embarked','Ticket','Cabin','Name']],axis = 1)
```

```
df1 = pd.concat([df,dummies], axis = 1)
df1
```

2.2.8.8.8.2 Ordinal Encoding

This is same as Label encoding, the key difference with ordinal encoding is that the numerical representation captures the ordinal relationship between the categories (there is a clear order or rank among the categories). This encoding assigns integer values to the categories based on their ordinal relationship.

This is crucial when dealing with features that have a meaningful order, as it helps the model understand and utilize the information about the relative positioning of the categories.

Ordinal encoding should be used with caution, as it introduces an artificial order that might not always reflect the true relationships between the categories.

Label Encoding is used to transform our dataset target variable to Numbers where each Label/class is represented by unique number. It is kind of similar to ordinal encoding however these numbers can't be ranked/ordered.

We must not use Label Encoding for the features. It is used only for target variable encoding.

In summary, use label encoding for nominal categorical variables (where there is no inherent order among categories like 'red', 'green', 'blue'), and use ordinal encoding for ordinal categorical variables (where there is a clear order or rank among categories like grads: 'A+', 'A', 'B+', 'B').

2.2.8.8.8.3 One-Hot Encoding:

One-Hot Encoding is another popular technique for treating categorical variables. It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature.

First, we will create some dummy variable

```
ports = pd.get_dummies(df.Embarked , prefix='Embarked')
ports.head()
```

	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1

Then we apply the One hot encoder

```
from sklearn.preprocessing import OneHotEncoder
```

```
prepared_df = pd.concat([df, pd.get_dummies(df['Embarked'],drop_first=True)], axis=1)
prepared_df.drop(['Embarked'], axis=1, inplace=True)
```

```
prepared_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Q	S
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	0	1
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	0	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	0	1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	0	1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	0	1

2.2.8.8.8.4 Dummy Encoding

Dummy coding scheme is similar to one-hot encoding. This categorical data encoding method transforms the categorical variable into a set of binary variables (also known as dummy variables). In the case of one-hot encoding, for N categories in a variable, it uses

N binary variables. The dummy encoding is a small improvement over one-hot-encoding. Dummy encoding uses N-1 features to represent N labels/categories.

Column	Code
A	100
B	010
C	001

One- Hot Coding

Column	Code
A	10
B	01
C	00

Dummy Code

```
import category_encoders as ce
import pandas as pd
data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad']})

#encode the data
data_encoded=pd.get_dummies(data=data,drop_first=True)
data_encoded
```

	City_Chennai	City_Delhi	City_Hyderabad	City_Mumbai
0	0	1	0	0
1	0	0	0	1
2	0	0	1	0
3	1	0	0	0
4	0	0	0	0
5	0	1	0	0
6	0	0	1	0

Here using *drop_first* argument, we are representing the first label Bangalore using 0.

2.2.8.8.8.5 Effect Encoding:

This encoding technique is also known as **Deviation Encoding** or **Sum Encoding.** Effect encoding is almost similar to dummy encoding, with a little difference. In dummy coding, we use 0 and 1 to represent the data but in effect encoding, we use three values i.e. 1,0, and -1.

The row containing only 0s in dummy encoding is encoded as -1 in effect encoding.

	intercept	City_0	City_1	City_2	City_3
0	1	1.0	0.0	0.0	0.0
1	1	0.0	1.0	0.0	0.0
2	1	0.0	0.0	1.0	0.0
3	1	0.0	0.0	0.0	1.0
4	1	-1.0	-1.0	-1.0	-1.0
5	1	1.0	0.0	0.0	0.0
6	1	0.0	0.0	1.0	0.0

2.2.8.8.8.6 HashEncoding:

Hashing is the process of converting of a string of characters into a unique hash value with applying a hash function. This process is quite useful as it can deal with a higher number of categorical data and its low memory usage.

We use hashing algorithms to perform hashing operations i.e to generate the hash value of an input.

Just like one-hot encoding, the Hash encoder represents categorical features using the new dimensions. Here, the user can fix the number of dimensions after transformation using ***n_component*** argument. That is a feature with 5 categories can be represented using N new features similarly, a feature with 100 categories can also be transformed using N new features.

By default, the Hashing encoder uses **the** md5 hashing algorithm but a user can pass any algorithm of his choice.

```
#Create the dataframe
data=pd.DataFrame({'Month':['January','April','March','April','Februay','June','July','June','September']})

#Create object for hash encoder
encoder=ce.HashingEncoder(cols='Month',n_components=6)

#Fit and Transform Data
encoder.fit_transform(data)
```

	col_0	col_1	col_2	col_3	col_4	col_5
0	0	0	0	0	1	0
1	0	0	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	1	0	0
4	0	0	0	1	0	0
5	0	1	0	0	0	0
6	1	0	0	0	0	0
7	0	1	0	0	0	0
8	0	0	0	0	1	0

Since Hashing transforms the data in lesser dimensions, it may lead to loss of information. Another issue faced by hashing encoder is the **collision.** Since here, a large number of features are depicted into lesser dimensions, hence multiple values can be represented by the same hash value, this is known as a collision.

It is great to try if the dataset has high cardinality features.

2.2.8.8.8.7 Binary Encoding:

Initially, categories are encoded as Integer and then converted into binary code, then the digits from that binary string are placed into separate columns.

for eg: for 7 : 1 1 1

This method is quite preferable when there is more categories. Imagine if you have 100 different categories. One hot encoding will create 100 different columns, But binary encoding only need 7 columns.

```
#Create the Dataframe
data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad','Mumbai','Agra']})

#Create object for binary encoding
encoder= ce.BinaryEncoder(cols=['City'])

#Fit and Transform Data
data_encoded=encoder.fit_transform(data['City'])
data_encoded
```

	City_0	City_1	City_2
0	0	0	1
1	0	1	0
2	0	1	1
3	1	0	0
4	1	0	1
5	0	0	1
6	0	1	1
7	0	1	0
8	1	1	0

2.2.8.8.8.8 8.Base N Encoding

For Binary encoding, the Base is 2 which means it converts the numerical values of a category into its respective Binary form. If you want to change the Base of encoding scheme you may use Base N encoder. In the case when categories are more and binary encoding is not able to handle the dimensionality then we can use a larger base such as 4 or 8.

```
#Create the dataframe
data=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad','Mumbai','Agra']})

#Create an object for Base N Encoding
encoder= ce.BaseNEncoder(cols=['City'],return_df=True,base=5)

#Fit and Transform Data
data_encoded=encoder.fit_transform(data)
data_encoded
```

	City_0	City_1
0	0	1
1	0	2
2	0	3
3	0	4
4	1	0
5	0	1
6	0	3
7	0	2
8	1	1

In the above example, we have used base 5 also known as the Quinary system. It is similar to the example of Binary encoding. While Binary encoding represents the same data by 3 new features the BaseN encoding uses only 2 new variables.

Hence BaseN encoding technique further reduces the number of features required to efficiently represent the data and improving memory usage. The default Base for Base N is 2 which is equivalent to Binary Encoding.

2.2.8.8.8.99. Target Encoding

Target encoding is a Baysian encoding technique.

Bayesian encoders use information from dependent/target variables to encode the categorical data.

In target encoding, we calculate the mean of the target variable for each category and replace the category variable with the mean value. In the case of the categorical target variables, the posterior probability of the target replaces each category.

```
#Create the Dataframe
data=pd.DataFrame({'class':['A,','B','C','B','C','A','A','A'],'Marks':[50,30,70,80,45,97,80,68]})

#Create target encoding object
encoder=ce.TargetEncoder(cols='class')

#Fit and Transform Train Data
encoder.fit_transform(data['class'],data['Marks'])
```

	class
0	63.048373
1	63.581489
2	63.936117
3	63.581489
4	63.936117
5	67.574421
6	67.574421
7	67.574421

We perform Target encoding for train data only and code the test data using results obtained from the training dataset. Although, a very efficient coding system, it has the following **issues** responsible for deteriorating the model performance-

- It can lead to target leakage or overfitting.
- we may face is the improper distribution of categories in train and test data.

2.2.8.8.8.10 10. Frequency/count Encoding

In this encoding method, we utilize the frequency of the categories as labels. In the cases where the frequency is related somewhat to the target variable, it helps the model understand and assign the weight in direct and inverse proportion, depending on the nature of the data.

```
#Create the Dataframe
df=pd.DataFrame({'City':['Delhi','Mumbai','Hyderabad','Chennai','Bangalore','Delhi','Hyderabad']})

#frequency of column
fe = df.groupby('City').size()/len(df)
df.loc[:,'encoded']= df['City'].map(fe)
df
```

	City	encoded
0	Delhi	0.285714
1	Mumbai	0.142857
2	Hyderabad	0.285714
3	Chennai	0.142857
4	Bangalore	0.142857
5	Delhi	0.285714
6	Hyderabad	0.285714

2.2.8.8.8.11 Mean Encoding

In this method, we will convert the categories into their mean values based on the output. This type of approach will be applicable where we have a lot of categorical variables for a particular column.

Created a DataFrame having two features named subjects and Target and we can see that here one of the features (SubjectName) is Categorical, so we have converted it into the numerical feature by applying Mean Encoding.

```
#Create the Dataframe
df=pd.DataFrame({'SubjectName':['s1','s2','s3','s1','s4','s3','s2','s1','s2','s4','s1'],
      'Target':[1,0,1,1,1,0,0,1,1,1,0]})

#groupby data with SubjectName with their mean according to their positive target value
Mean_encoded_subject = df.groupby(['SubjectName'])['Target'].mean().to_dict()
df['SubjectName'] =  df['SubjectName'].map(Mean_encoded_subject)
df
```

	SubjectName	Target
0	0.750000	1
1	0.333333	0
2	0.500000	1
3	0.750000	1
4	1.000000	1
5	0.500000	0
6	0.333333	0
7	0.750000	1
8	0.333333	1
9	1.000000	1
10	0.750000	0

2.2.8.8.9 Power Transformation (Yeo-Johnson Transformation):

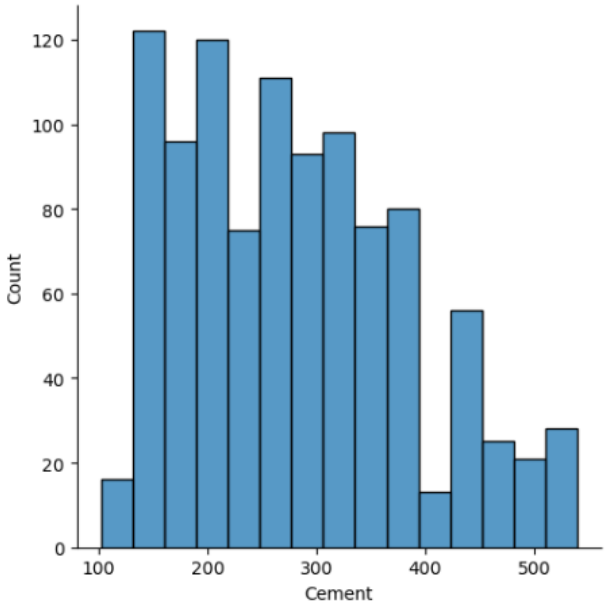
Power transformation is a technique used in statistics and data analysis to stabilize variance, make data more normally distributed, or handle non-linear relationships. It involves applying mathematical transformations to the data in order to achieve the desired

objectives. Similar to the Box-Cox transformation, the Yeo-Johnson transformation is a family of power transformations that can handle both positive and negative data values. It allows for more flexibility in transforming data compared to the Box-Cox method.

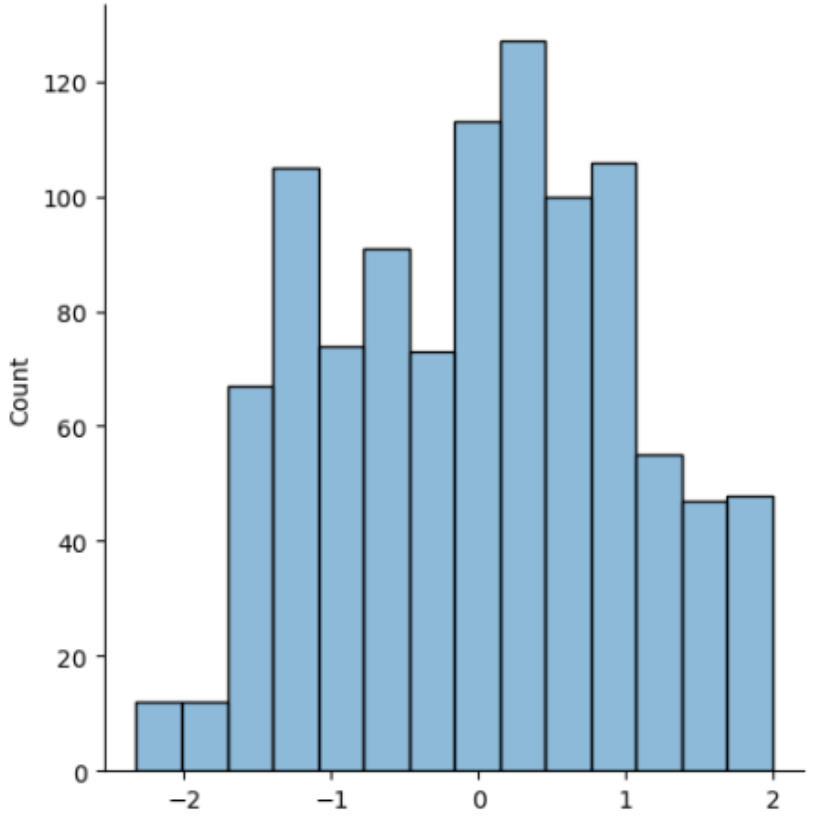
	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

In [29]:
```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
```

In [30]:
```
sns.displot(dfx['Cement'])
plt.show()
```



```
In [35]: sns.displot(pt.fit_transform(dfx[['Cement']]))
         plt.show()
```



2.2.9 Data Joining and Merging:

Combine multiple datasets based on common columns.

Data Joining and Merging Functions:

2.2.9.1 merge

The merge() method updates the content of two DataFrame by merging them together, using the specified method(s).

	key	Name	Age
0	K0	Jai	27
1	K1	Princi	24
2	K2	Gaurav	22
3	K3	Anuj	32

```
In [118]: df1
```

Out[118]:

	key	Address	Qualification
0	K0	Nagpur	Btech
1	K1	Kanpur	B.A
2	K2	Allahabad	Bcom
3	K3	Kannuaj	B.hons

```
In [119]: m = pd.merge(df,df1, on= 'key')
          m
```

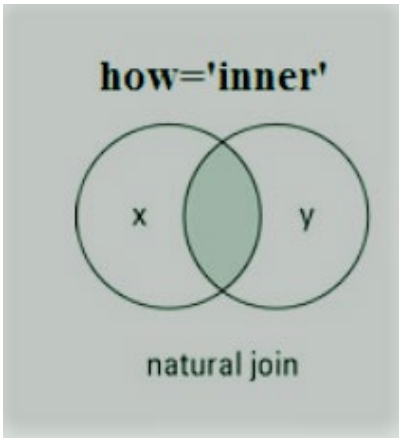
Out[119]:

	key	Name	Age	Address	Qualification
0	K0	Jai	27	Nagpur	Btech
1	K1	Princi	24	Kanpur	B.A
2	K2	Gaurav	22	Allahabad	Bcom
3	K3	Anuj	32	Kannuaj	B.hons

2.2.9.2 join

2.2.9.2.1 Pandas Inner Join

Inner join is the most common type of join you'll be working with. It returns a Dataframe with only those rows that have common characteristics. This is similar to the intersection of two sets.



Out[120]:

	id	val1
0	1	a
1	2	b
2	10	c
3	12	d

In [121]:
```
b
```

Out[121]:

	id	val1
0	1	p
1	2	q
2	9	r
3	8	s

In [124]:
```
e = pd.merge(a,b,on = 'id',how= 'inner')
e
```

Out[124]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q

2.2.9.2.2 Pandas Left Join:

With a left outer join, all the records from the first Dataframe will be displayed, irrespective of whether the keys in the first Dataframe can be found in the second Dataframe. Whereas, for the second Dataframe, only the records with the keys in the second Dataframe that can be found in the first Dataframe will be displayed.

In [125]:
```
lo = pd.merge(a,b,on = 'id',how= 'left')
lo
```

Out[125]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN

2.2.9.2.3 Pandas Right Outer Join:

For a right join, all the records from the second Dataframe will be displayed. However, only the records with the keys in the first Dataframe that can be found in the second Dataframe will be displayed.

```
In [126]: ro = pd.merge(a,b,on = 'id',how= 'right')
          ro
```

Out[126]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	9	NaN	r
3	8	NaN	s

2.2.9.2.4 Pandas Full Outer Join:

A full outer join returns all the rows from the left Dataframe, and all the rows from the right Dataframe, and matches up rows where possible, with NaNs elsewhere. But if the Dataframe is complete, then we get the same output.

```
In [127]: o = pd.merge(a,b,on = 'id',how= 'outer')
          o
```

Out[127]:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN
4	9	NaN	r
5	8	NaN	s

2.2.9.3 concat:

The pandas.concat() function does all the heavy lifting of performing concatenation operations along with an axis of Pandas objects while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

```
In [128]: a
```

Out[128]:

	level
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

```
In [129]: b
```

Out[129]:

	power
0	10
1	20
2	30
3	40
4	50
5	60
6	70
7	80
8	90
9	100

```
In [130]: c = pd.concat([a,b],axis=1)
          c
```

Out[130]:

	level	power
0	1	10
1	2	20
2	3	30
3	4	40
4	5	50
5	6	60
6	7	70
7	8	80
8	9	90
9	10	100

2.2.10 Data Reshaping Functions:

2.2.10.1 Pivot

pandas.pivot(index, columns, values) function produces pivot table based on 3 columns of the DataFrame. Uses unique values from index / columns and fills with values.

Out[131]:

	A	B	C
0	John	Masters	27
1	Boby	Graduate	23
2	Mina	Graduate	21

In [132]:
```
print(students.pivot('A','B','C'))
```

```
B      Graduate  Masters
A
Boby       23.0      NaN
John        NaN     27.0
Mina       21.0      NaN
```

In [133]:
```
students.pivot(index ='A', columns ='B', values =['C', 'A'])
```

Out[133]:

	C		A	
B	Graduate	Masters	Graduate	Masters
A				
Boby	23	NaN	Boby	NaN
John	NaN	27	NaN	John
Mina	21	NaN	Mina	NaN

2.2.10.2 Melt

melt() function is useful to convert a DataFrame into a format where one or more columns are identifier variables, while all other columns, considered measured variables, are unpivoted to the row axis, leaving just two non-identifier columns, variable and value.

In [137]:
```
df2 = pd.melt(students, id_vars =['B'], value_vars =['A', 'C'], ignore_index= True)
print(df2)
```

```
          B variable value
0   Masters        A  John
1  Graduate        A  Boby
2  Graduate        A  Mina
3   Masters        C    27
4  Graduate        C    23
5  Graduate        C    21
```

2.2.10.3 Stack:

Stack method works with the MultiIndex objects in DataFrame, it returning a DataFrame with an index with a new inner-most level of row labels. It changes the wide table to a long table.

Out[140]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

```
In [147]: stacked = df4.stack()
          stacked.head(10)

Out[147]: 0  Name          Avery Bradley
             Team          Boston Celtics
             Number                   0.0
             Position                  PG
             Age                     25.0
             Height                   6-2
             Weight                 180.0
             College                Texas
             Salary           7730337.0
          1  Name            Jae Crowder
          dtype: object
```

2.2.10.4 Unstack

unstack is similar to stack method, It also works with multi-index objects in dataframe, producing a reshaped DataFrame with a new inner-most level of column labels.

```
In [147]: stacked = df4.stack()
          stacked.head(10)

Out[147]: 0  Name          Avery Bradley
             Team          Boston Celtics
             Number                   0.0
             Position                  PG
             Age                     25.0
             Height                   6-2
             Weight                 180.0
             College                Texas
             Salary           7730337.0
          1  Name            Jae Crowder
          dtype: object

In [148]: unstacked = stacked.unstack()
          unstacked.head(10)
```

Out[148]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

2.2.10.5 transpose (or .T)

DataFrame.transpose() function transpose index and columns of the dataframe. It reflect the DataFrame over its main diagonal by writing rows as columns and vice-versa.

	Weight	Name	Age
2010-10-09 08:45:00	45	Sam	14
2010-10-09 09:45:00	88	Andrea	25
2010-10-09 10:45:00	56	Alex	55
2010-10-09 11:45:00	15	Robin	8
2010-10-09 12:45:00	71	Kia	21

```
In [151]: t = df2.transpose()
          t
```

Out[151]:

	2010-10-09 08:45:00	2010-10-09 09:45:00	2010-10-09 10:45:00	2010-10-09 11:45:00	2010-10-09 12:45:00
Weight	45	88	56	15	71
Name	Sam	Andrea	Alex	Robin	Kia
Age	14	25	55	8	21

2.2.10.6 reindex

Reindexing is used to change the index of the rows and columns of the DataFrame. We can reindex the single or multiple rows by using the reindex() method. Default values in the new index are assigned NaN if it is not present in the DataFrame.

Out[178]:

	a	b	c	d	e
A	0.101992	0.597419	0.111035	0.219574	0.428337
B	0.686775	0.814983	0.448665	0.670447	0.572794
C	0.553909	0.204193	0.106599	0.170051	0.714756
D	0.280149	0.631719	0.244786	0.788333	0.133868
E	0.898927	0.715223	0.276094	0.257453	0.161398

```
In [179]: new_index =['U', 'A', 'B', 'C', 'Z']
          df1.reindex(new_index,fill_value='0.5')
```

Out[179]:

	a	b	c	d	e
U	0.5	0.5	0.5	0.5	0.5
A	0.101992	0.597419	0.111035	0.219574	0.428337
B	0.686775	0.814983	0.448665	0.670447	0.572794
C	0.553909	0.204193	0.106599	0.170051	0.714756
Z	0.5	0.5	0.5	0.5	0.5

2.2.10.7 pivot_table

The pivot_table() function is used to create a spreadsheet-style pivot table as a DataFrame. The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame..

Out[180]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
...	...	...	...	...	...	...	...	...	...
453	Shelvin Mack	Utah Jazz	8.0	PG	26.0	6-3	203.0	Butler	2433333.0
454	Raul Neto	Utah Jazz	25.0	PG	24.0	6-1	179.0	NaN	900000.0
455	Tibor Pleiss	Utah Jazz	21.0	C	26.0	7-3	256.0	NaN	2900000.0
456	Jeff Withey	Utah Jazz	24.0	C	26.0	7-0	231.0	Kansas	947276.0
457	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

458 rows × 9 columns

In [185]:
```
changed =pd.pivot_table(df4,index = ['Team'],aggfunc={'Salary': np.sum,'Age':np.mean,'Weight':np.mean})
changed
```

Out[185]:

Team	Age	Salary	Weight
Atlanta Hawks	28.200000	72902950.0	221.266667
Boston Celtics	24.733333	58541068.0	219.466667
Brooklyn Nets	25.600000	52528475.0	215.600000
Charlotte Hornets	26.133333	78340920.0	220.400000
Chicago Bulls	27.400000	86783378.0	218.933333
Cleveland Cavaliers	29.533333	106968689.0	227.866667
Dallas Mavericks	29.733333	71198732.0	227.000000
Denver Nuggets	25.733333	60121930.0	217.533333
Detroit Pistons	26.200000	67168263.0	222.200000
Golden State Warriors	27.666667	88868997.0	224.600000
Houston Rockets	26.866667	75283021.0	220.333333
Indiana Pacers	26.400000	66751826.0	222.266667
Los Angeles Clippers	29.466667	94854640.0	219.733333
Los Angeles Lakers	27.533333	71770431.0	227.066667
Memphis Grizzlies	28.388889	76550880.0	218.000000
Miami Heat	28.933333	82515673.0	218.400000
Milwaukee Bucks	24.562500	69603517.0	224.062500
Minnesota Timberwolves	26.357143	59709697.0	226.642857
New Orleans Pelicans	26.894737	82750774.0	221.000000
New York Knicks	27.000000	73303898.0	223.625000
Oklahoma City Thunder	27.066667	93765298.0	229.400000
Orlando Magic	25.071429	60161470.0	213.357143
Philadelphia 76ers	24.600000	30992894.0	222.133333
Phoenix Suns	25.866667	63445135.0	218.600000
Portland Trail Blazers	25.066667	48301818.0	218.600000
Sacramento Kings	26.800000	71683666.0	221.333333
San Antonio Spurs	31.600000	84442733.0	223.933333
Toronto Raptors	26.133333	71117611.0	221.800000
Utah Jazz	24.466667	63060091.0	220.000000
Washington Wizards	27.866667	76328636.0	219.000000

2.2.11 Data Splitting:

Divide the dataset into training and testing sets for analysis or modeling.

Data Splitting and Feature Engineering Functions:

2.2.11.1 train_test_split

The train_test_split() method is used to split our data into train and test sets.

```
In [33]: 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=2)
         X_train
```

Out[33]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
128	401.8	94.7	0.0	147.4	11.4	946.8	852.1	28
365	214.9	53.8	121.9	155.6	9.6	1014.3	780.6	14
480	446.0	24.0	79.0	162.0	11.6	967.0	712.0	7
814	310.0	0.0	0.0	192.0	0.0	970.0	850.0	360
169	425.0	106.3	0.0	153.5	16.5	852.1	887.1	91
...	...	...	...	...	...	...	...	...
360	218.2	54.6	123.8	140.8	11.9	1075.7	792.7	14
466	190.3	0.0	125.2	166.6	9.9	1079.0	798.9	100
299	290.4	0.0	96.2	168.1	9.4	961.2	865.0	3
493	387.0	20.0	94.0	157.0	11.6	938.0	845.0	7
527	359.0	19.0	141.0	154.0	10.9	942.0	801.0	7

824 rows × 8 columns


```
In [33]: 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=2)
         X_train
```

Out[33]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
128	401.8	94.7	0.0	147.4	11.4	946.8	852.1	28
365	214.9	53.8	121.9	155.6	9.6	1014.3	780.6	14
480	446.0	24.0	79.0	162.0	11.6	967.0	712.0	7
814	310.0	0.0	0.0	192.0	0.0	970.0	850.0	360
169	425.0	106.3	0.0	153.5	16.5	852.1	887.1	91
...	...	...	...	...	...	...	...	...
360	218.2	54.6	123.8	140.8	11.9	1075.7	792.7	14
466	190.3	0.0	125.2	166.6	9.9	1079.0	798.9	100
299	290.4	0.0	96.2	168.1	9.4	961.2	865.0	3
493	387.0	20.0	94.0	157.0	11.6	938.0	845.0	7
527	359.0	19.0	141.0	154.0	10.9	942.0	801.0	7

In [34]:
```
X_test
```

Out[34]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
456	194.7	0.0	100.5	170.2	7.5	998.0	901.8	56
988	153.6	144.2	112.3	220.1	10.1	923.2	657.9	28
809	252.0	0.0	0.0	185.0	0.0	1111.0	784.0	28
581	181.9	272.8	0.0	185.7	0.0	1012.4	714.3	7
549	333.0	0.0	0.0	192.0	0.0	931.2	842.6	90
...	...	...	...	...	...	...	...	...
839	153.0	239.0	0.0	200.0	6.0	1002.0	684.0	28
455	213.5	0.0	174.2	159.2	11.7	1043.6	771.9	56
691	212.0	141.3	0.0	203.5	0.0	973.4	750.0	3
934	184.0	86.0	190.0	213.0	6.0	923.0	623.0	28
172	425.0	106.3	0.0	153.5	16.5	852.1	887.1	91

206 rows × 8 columns

In [35]:
```
y_train
```

Out[35]:

	Strength
128	68.50
365	38.60
480	39.30
814	38.11
169	65.20
...	...
360	35.96
466	33.56
299	22.50
493	41.67
527	35.75

824 rows × 1 columns

In [36]:
```
y_test
```

Out[36]:

	Strength
456	43.39
988	16.50
809	19.69
581	12.37
549	41.68
...	...
839	26.86
455	51.26
691	6.81
934	22.93
172	65.20

206 rows × 1 columns

2.2.11.2 pd.to_datetime

When a csv file is imported and a Data Frame is made, the Date time objects in the file are read as a string object rather a Date Time object and hence it's very tough to perform operations like Time difference on a string rather a Date Time object. Pandas to_datetime() method helps to convert string Date time into Python Date time object.

Out[37]:

	Date	Time
0	8/6/1993	12:42 PM
1	3/31/1996	6:53 AM
2	4/23/1993	11:17 AM
3	3/4/2005	1:00 PM
4	1/24/1998	4:47 PM

In [38]:
```
dateis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Date    1000 non-null   object
 1   Time    1000 non-null   object
dtypes: object(2)
memory usage: 15.8+ KB
```

In [39]:
```
dateis['Date'] = pd.to_datetime(dateis['Date'])
dateis.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   Date    1000 non-null   datetime64[ns]
 1   Time    1000 non-null   object        
dtypes: datetime64[ns](1), object(1)
memory usage: 15.8+ KB
```

2.2.11.3 pd.to_numeric

pandas.to_numeric() is one of the general functions in Pandas which is used to convert argument to a numeric type.

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0

```
In [ ]: df = pd.read_csv("nba.csv")
        ser = pd.Series(df['Number']).head(10)
        pd.to_numeric(ser, downcast ='signed')
```

```
0     0
1    99
2    30
3    28
4     8
5    90
6    55
7    41
8    12
9    36
Name: Number, dtype: int8
```

2.2.11.4 pd.cut:

Pandas cut() function is used to separate the array elements into different bins . The cut function is mainly used to perform statistical analysis on scalar data.

In [5]:
```
import pandas as pd
import numpy as np
import random
info_nums = pd.DataFrame({'num': np.random.randint(1, 10, 7)})
print(info_nums)
info_nums['nums_labels'] = pd.cut(x=info_nums['num'], bins=[1, 7, 10], labels=['Lows', 'Highs'], right=False)
print(info_nums)
print(info_nums['nums_labels'].unique())
```

```
   num
0    7
1    8
2    4
3    7
4    9
5    4
6    8
   num nums_labels
0    7       Highs
1    8       Highs
2    4        Lows
3    7       Highs
4    9       Highs
5    4        Lows
6    8       Highs
['Highs', 'Lows']
Categories (2, object): ['Lows' < 'Highs']
```

2.2.11.5 pd.qcut

qcut (quantile-cut) differs from cut in the sense that, in qcut, the number of elements in each bin will be roughly the same, but this will come at the cost of differently sized interval widths. On the other hand, in cut, the bin edges were equal sized (when we specified bins=3) with uneven number of elements in each bin or group.

:
```
import pandas as pd
import numpy as np
info_nums = pd.DataFrame({'num': np.random.randint(1, 10, 7)})
print(info_nums)
info_nums['nums_labels'] = pd.qcut(x=info_nums['num'], q=4, duplicates='drop')
print(info_nums)
print(info_nums['nums_labels'].unique())
```

```
   num
0    3
1    4
2    3
3    8
4    6
5    3
6    9
   num   nums_labels
0    3  (2.999, 4.0]
1    4  (2.999, 4.0]
2    3  (2.999, 4.0]
3    8    (7.0, 9.0]
4    6    (4.0, 7.0]
5    3  (2.999, 4.0]
6    9    (7.0, 9.0]
[(2.999, 4.0], (7.0, 9.0], (4.0, 7.0]]
Categories (3, interval[float64, right]): [(2.999, 4.0] < (4.0, 7.0] < (7.0, 9.0]]
```

2.2.11.6 pd.get_dummies

The get_dummies() function is used to convert categorical variable into dummy/indicator variables.The function iterates over the object that is passed and checks if the element at the particular index matches the column heading.If it does, it encodes it as a 1.Otherwise, it assigns it a 0.

Out[27]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

In [29]:
```
dfx = pd.get_dummies(df['Sex'])
dfx
```

Out[29]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

891 rows × 2 columns

2.2.12 Data Visualization:

Create plots and charts to visualize data patterns and relationships.

2.2.12.1 Data Visualization Libraries

- Matplotlib
- Seaborn
- ggplot2
- Plotly
- D3.js
- Tableau
- Power BI
- Plotly Dash
- Altair
- Bokeh
- Folium
- NetworkX
- Yellowbrick
- 3D Visualization in Plotly
- Panel Visualization in Plotly
- Plotly Dash Dashboarding