**Exploratory Data Analysis (EDA)**

Exploratory data analysis is one of the basic and essential steps of a data science project. A data scientist involves almost 70% of his work in doing the EDA of the dataset.

**What is Exploratory Data Analysis (EDA)?**

Exploratory Data Analysis (EDA) is a crucial initial step in data science projects. It involves analyzing and visualizing data to understand its key characteristics, uncover patterns, and identify relationships between variables refers to the method of studying and exploring record sets to apprehend their predominant traits, discover patterns, locate outliers, and identify relationships between variables. EDA is normally carried out as a preliminary step before undertaking extra formal statistical analyses or modeling.

**Key aspects of EDA include:**

* **Distribution of Data**: Examining the distribution of data points to understand their range, central tendencies (mean, median), and dispersion (variance, standard deviation).
* **Graphical Representations**: Utilizing charts such as histograms, box plots, scatter plots, and bar charts to visualize relationships within the data and distributions of variables.
* **Outlier Detection**: Identifying unusual values that deviate from other data points. Outliers can influence statistical analyses and might indicate data entry errors or unique cases.
* **Correlation Analysis**: Checking the relationships between variables to understand how they might affect each other. This includes computing correlation coefficients and creating correlation matrices.
* **Handling Missing Values**: Detecting and deciding how to address missing data points, whether by imputation or removal, depending on their impact and the amount of missing data.
* **Summary Statistics:** Calculating key statistics that provide insight into data trends and nuances.
* **Testing Assumptions**: Many statistical tests and models assume the data meet certain conditions (like normality or homoscedasticity). EDA helps verify these assumptions.

**Why Exploratory Data Analysis is Important?**

Exploratory Data Analysis (EDA) is important for several reasons, especially in the context of data science and statistical modeling. Here are some of the key reasons why EDA is a critical step in the data analysis process:

1. **Understanding Data Structures**: EDA helps in getting familiar with the dataset, understanding the number of features, the type of data in each feature, and the distribution of data points. This understanding is crucial for selecting appropriate analysis or prediction techniques.
2. **Identifying Patterns and Relationships**: Through visualizations and statistical summaries, EDA can reveal hidden patterns and intrinsic relationships between variables. These insights can guide further analysis and enable more effective feature engineering and model building.
3. **Detecting Anomalies and Outliers**: EDA is essential for identifying errors or unusual data points that may adversely affect the results of your analysis. Detecting these early can prevent costly mistakes in predictive modeling and analysis.
4. **Testing Assumptions**: Many statistical models assume that data follow a certain distribution or that variables are independent. EDA involves checking these assumptions. If the assumptions do not hold, the conclusions drawn from the model could be invalid.
5. **Informing Feature Selection and Engineering**: Insights gained from EDA can inform which features are most relevant to include in a model and how to transform them (scaling, encoding) to improve model performance.
6. **Optimizing Model Design**: By understanding the data’s characteristics, analysts can choose appropriate modeling techniques, decide on the complexity of the model, and better tune model parameters.
7. **Facilitating Data Cleaning**: EDA helps in spotting missing values and errors in the data, which are critical to address before further analysis to improve data quality and integrity.
8. **Enhancing Communication**: Visual and statistical summaries from EDA can make it easier to communicate findings and convince others of the validity of your conclusions, particularly when explaining data-driven insights to stakeholders without technical backgrounds.

**Types of Exploratory Data Analysis**

EDA, or Exploratory Data Analysis, refers back to the method of analyzing and analyzing information units to uncover styles, pick out relationships, and gain insights. There are various sorts of EDA strategies that can be hired relying on the nature of the records and the desires of the evaluation. Depending on the number of columns we are analyzing we can divide EDA into three types: [Univariate, bivariate and multivariate](https://www.geeksforgeeks.org/univariate-bivariate-and-multivariate-data-and-its-analysis/).

**1. Univariate Analysis**

Univariate analysis focuses on a single variable to understand its internal structure. It is primarily concerned with describing the data and finding patterns existing in a single feature. This sort of evaluation makes a speciality of analyzing character variables inside the records set. It involves summarizing and visualizing a unmarried variable at a time to understand its distribution, relevant tendency, unfold, and different applicable records. Common techniques include:

* **Histograms**: Used to visualize the distribution of a variable.
* **Box plots**: Useful for detecting outliers and understanding the spread and skewness of the data.
* **Bar charts**: Employed for categorical data to show the frequency of each category.
* **Summary statistics**: Calculations like mean, median, mode, variance, and standard deviation that describe the central tendency and dispersion of the data.

**2. Bivariate Analysis**

Bivariate evaluation involves exploring the connection between variables. It enables find associations, correlations, and dependencies between pairs of variables. Bivariate analysis is a crucial form of exploratory data analysis that examines the relationship between two variables. Some key techniques used in bivariate analysis:

* **Scatter Plots:**These are one of the most common tools used in bivariate analysis. A scatter plot helps visualize the relationship between two continuous variables.
* **Correlation Coefficient**: This statistical measure (often Pearson’s correlation coefficient for linear relationships) quantifies the degree to which two variables are related.
* **Cross-tabulation**: Also known as contingency tables, cross-tabulation is used to analyze the relationship between two categorical variables. It shows the frequency distribution of categories of one variable in rows and the other in columns, which helps in understanding the relationship between the two variables.
* **Line Graphs**: In the context of time series data, line graphs can be used to compare two variables over time. This helps in identifying trends, cycles, or patterns that emerge in the interaction of the variables over the specified period.
* **Covariance**: Covariance is a measure used to determine how much two random variables change together. However, it is sensitive to the scale of the variables, so it’s often supplemented by the correlation coefficient for a more standardized assessment of the relationship.

**3. Multivariate Analysis**

Multivariate analysis examines the relationships between two or more variables in the dataset. It aims to understand how variables interact with one another, which is crucial for most statistical modeling techniques. Techniques include:

* **Pair plots**: Visualize relationships across several variables simultaneously to capture a comprehensive view of potential interactions.
* **Principal Component Analysis (PCA)**: A dimensionality reduction technique used to reduce the dimensionality of large datasets, while preserving as much variance as possible.

**Specialized EDA Techniques**

In addition to univariate and multivariate analysis, there are specialized EDA techniques tailored for specific types of data or analysis needs:

* **Spatial Analysis**: For geographical data, using maps and spatial plotting to understand the geographical distribution of variables.
* **Text Analysis**: Involves techniques like word clouds, frequency distributions, and sentiment analysis to explore text data.
* **Time Series Analysis:** This type of analysis is mainly applied to statistics sets that have a temporal component. Time collection evaluation entails inspecting and modeling styles, traits, and seasonality inside the statistics through the years. Techniques like line plots, autocorrelation analysis, transferring averages, and ARIMA (AutoRegressive Integrated Moving Average) fashions are generally utilized in time series analysis.

**Tools for Performing Exploratory Data Analysis**

Exploratory Data Analysis (EDA) can be effectively performed using a variety of tools and software, each offering unique features suitable for handling different types of data and analysis requirements.

**1. Python Libraries**

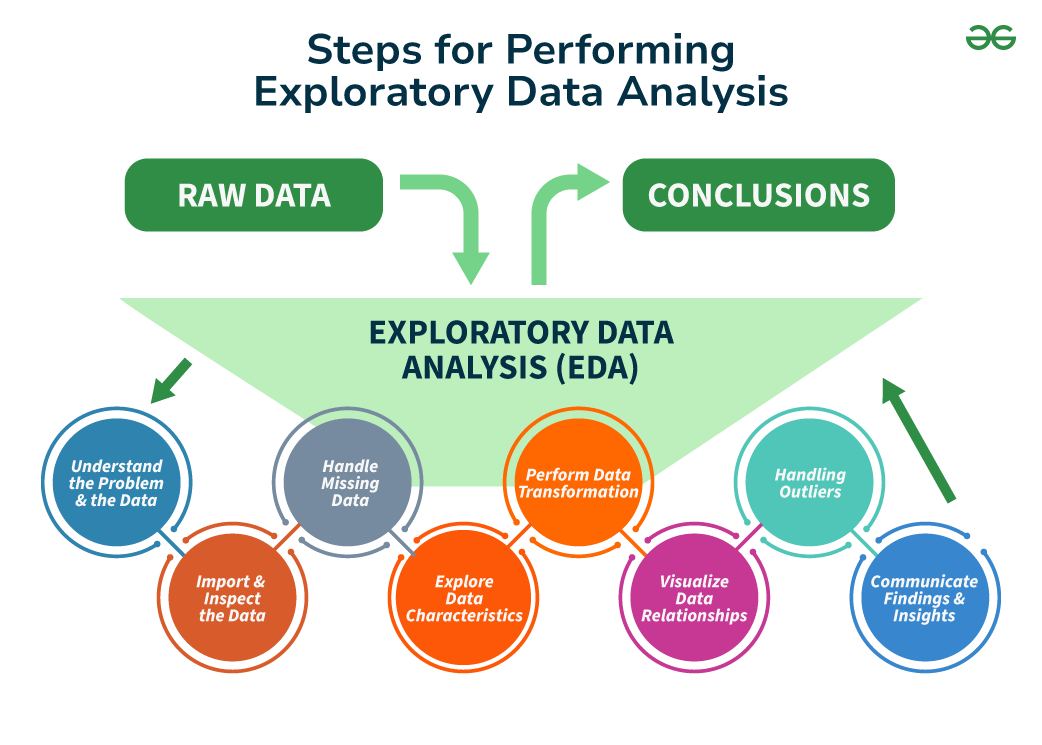
* **Pandas**: Provides extensive functions for data manipulation and analysis, including data structure handling and time series functionality.
* **Matplotlib**: A plotting library for creating static, interactive, and animated visualizations in Python.
* **Seaborn**: Built on top of Matplotlib, it provides a high-level interface for drawing attractive and informative statistical graphics.
* **Plotly**: An interactive graphing library for making interactive plots and offers more sophisticated visualization capabilities.

**2. R Packages**

* **ggplot2**: Part of the tidyverse, it’s a powerful tool for making complex plots from data in a data frame.
* **dplyr**: A grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges.
* **tidyr**: Helps to tidy your data. Tidying your data means storing it in a consistent form that matches the semantics of the dataset with the way it is stored.

**Steps for Performing Exploratory Data Analysis**

Performing Exploratory Data Analysis (EDA) involves a series of steps designed to help you understand the data you’re working with, uncover underlying patterns, identify anomalies, test hypotheses, and ensure the data is clean and suitable for further analysis.



**Step 1: Understand the Problem and the Data**

The first step in any information evaluation project is to sincerely apprehend the trouble you are trying to resolve and the statistics you have at your disposal. This entails asking questions consisting of:

* What is the commercial enterprise goal or research question you are trying to address?
* What are the variables inside the information, and what do they mean?
* What are the data sorts (numerical, categorical, textual content, etc.) ?
* Is there any known information on first-class troubles or obstacles?
* Are there any relevant area-unique issues or constraints?

By thoroughly knowing the problem and the information, you can better formulate your evaluation technique and avoid making incorrect assumptions or drawing misguided conclusions. It is also vital to contain situations and remember specialists or stakeholders to this degree to ensure you have complete know-how of the context and requirements.

**Step 2: Import and Inspect the Data**

Once you have clean expertise of the problem and the information, the following step is to import the data into your evaluation environment (e.g., Python, R, or a spreadsheet program). During this step, looking into the statistics is critical to gain initial know-how of its structure, variable kinds, and capability issues.

Here are a few obligations you could carry out at this stage:

* Load the facts into your analysis environment, ensuring that the facts are imported efficiently and without errors or truncations.
* Examine the size of the facts (variety of rows and columns) to experience its length and complexity.
* Check for missing values and their distribution across variables, as missing information can notably affect the quality and reliability of your evaluation.
* Identify facts sorts and formats for each variable, as these records may be necessary for the following facts manipulation and evaluation steps.
* Look for any apparent errors or inconsistencies in the information, such as invalid values, mismatched units, or outliers, that can indicate exceptional issues with information.

**Step 3: Handle Missing Data**

Missing records is a joint project in many datasets, and it can significantly impact the quality and reliability of your evaluation. During the EDA method, it’s critical to pick out and deal with lacking information as it should be, as ignoring or mishandling lacking data can result in biased or misleading outcomes.

Here are some techniques you could use to handle missing statistics:

* **Understand the styles and capacity reasons for missing statistics**: Is the information lacking entirely at random (MCAR), lacking at random (MAR), or lacking not at random (MNAR)? Understanding the underlying mechanisms can inform the proper method for handling missing information.
* **Decide whether to eliminate observations with lacking values (listwise deletion) or attribute (fill in) missing values**: Removing observations with missing values can result in a loss of statistics and potentially biased outcomes, specifically if the lacking statistics are not MCAR. Imputing missing values can assist in preserving treasured facts. However, the imputation approach needs to be chosen cautiously.
* **Use suitable imputation strategies**, such as mean/median imputation, regression imputation, a couple of imputations, or device-getting-to-know-based imputation methods like k-nearest associates (KNN) or selection trees. The preference for the imputation technique has to be primarily based on the characteristics of the information and the assumptions underlying every method.
* **Consider the effect of lacking information**: Even after imputation, lacking facts can introduce uncertainty and bias. It is important to acknowledge those limitations and interpret your outcomes with warning.

Handling missing information nicely can improve the accuracy and reliability of your evaluation and save you biased or deceptive conclusions. It is likewise vital to record the techniques used to address missing facts and the motive in the back of your selections.

**Step 4: Explore Data Characteristics**

After addressing the facts that are lacking, the next step within the EDA technique is to explore the traits of your statistics. This entails examining your variables’ distribution, crucial tendency, and variability and identifying any ability outliers or anomalies. Understanding the characteristics of your information is critical in deciding on appropriate analytical techniques, figuring out capability information first-rate troubles, and gaining insights that may tell subsequent evaluation and modeling decisions.

**Calculate summary facts** (suggest, median, mode, preferred deviation, skewness, kurtosis, and many others.) for numerical variables: These facts provide a concise assessment of the distribution and critical tendency of each variable, aiding in the identification of ability issues or deviations from expected patterns.

**Step 5: Perform Data Transformation**

Data transformation is a critical step within the EDA process because it enables you to prepare your statistics for similar evaluation and modeling. Depending on the traits of your information and the necessities of your analysis, you may need to carry out various ameliorations to ensure that your records are in the most appropriate layout.

Here are a few common records transformation strategies:

* Scaling or normalizing numerical variables to a standard variety (e.g., [min-max scaling, standardization](https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/))
* Encoding categorical variables to be used in machine mastering fashions (e.g., one-warm encoding, label encoding)
* Applying mathematical differences to numerical variables (e.g., logarithmic, square root) to correct for skewness or non-linearity
* Creating derived variables or capabilities primarily based on current variables (e.g., calculating ratios, combining variables)
* Aggregating or grouping records mainly based on unique variables or situations

By accurately transforming your information, you could ensure that your evaluation and modeling strategies are implemented successfully and that your results are reliable and meaningful.

**Step 6: Visualize Data Relationships**

Visualization is an effective tool in the EDA manner, as it allows to discover relationships between variables and become aware of styles or trends that may not immediately be apparent from summary statistics or numerical outputs. To visualize data relationships, explore univariate, bivariate, and multivariate analysis.

* Create frequency tables, bar plots, and pie charts for express variables: These visualizations can help you apprehend the distribution of classes and discover any ability imbalances or unusual patterns.
* Generate histograms, container plots, violin plots, and density plots to visualize the distribution of numerical variables. These visualizations can screen critical information about the form, unfold, and ability outliers within the statistics.
* Examine the correlation or association among variables using scatter plots, correlation matrices, or statistical assessments like Pearson’s correlation coefficient or Spearman’s rank correlation: Understanding the relationships between variables can tell characteristic choice, dimensionality discount, and modeling choices.

**Step 7: Handling Outliers**

An [Outlier](https://www.geeksforgeeks.org/machine-learning-outlier/) is a data item/object that deviates significantly from the rest of the (so-called normal)objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect outliers, and the removal process of these outliers from the dataframe is the same as removing a data item from the panda’s dataframe.

Identify and inspect capability outliers through the usage of strategies like the [interquartile range (IQR)](https://www.geeksforgeeks.org/interquartile-range-to-detect-outliers-in-data/),[Z-scores](https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/), or area-specific regulations: Outliers can considerably impact the results of statistical analyses and gadget studying fashions, so it’s essential to perceive and take care of them as it should be.

**Step 8: Communicate Findings and Insights**

The final step in the EDA technique is effectively discussing your findings and insights. This includes summarizing your evaluation, highlighting fundamental discoveries, and imparting your outcomes cleanly and compellingly.

Here are a few hints for effective verbal exchange:

* Clearly state the targets and scope of your analysis
* Provide context and heritage data to assist others in apprehending your approach
* Use visualizations and photos to guide your findings and make them more reachable
* Highlight critical insights, patterns, or anomalies discovered for the duration of the EDA manner
* Discuss any barriers or caveats related to your analysis
* Suggest ability next steps or areas for additional investigation

Effective conversation is critical for ensuring that your EDA efforts have a meaningful impact and that your insights are understood and acted upon with the aid of stakeholders.

**Data cleaning: handling missing values, outliers, and noise.**

**Data cleaning** is one of the important parts of machine learning. It plays a significant part in building a model. In this article, we’ll understand Data cleaning, its significance and Python implementation.

**What is Data Cleaning?**

Data cleaning is a crucial step in the machine learning (ML) pipeline, as it involves identifying and removing any missing, duplicate, or irrelevant data. The goal of data cleaning is to ensure that the data is accurate, consistent, and free of errors, as incorrect or inconsistent data can negatively impact the performance of the ML model. Professional data scientists usually invest a very large portion of their time in this step because of the belief that **“Better data beats fancier algorithms”**.

Data cleaning, also known as **data cleansing**or **data preprocessing**, is a crucial step in the data science pipeline that involves identifying and correcting or removing errors, inconsistencies, and inaccuracies in the data to improve its quality and usability. Data cleaning is essential because raw data is often noisy, incomplete, and inconsistent, which can negatively impact the accuracy and reliability of the insights derived from it.

**Why is Data Cleaning Important?**

Data cleansing is a crucial step in the data preparation process, playing an important role in ensuring the accuracy, reliability, and overall quality of a dataset.

For decision-making, the integrity of the conclusions drawn heavily relies on the cleanliness of the underlying data. Without proper data cleaning, inaccuracies, outliers, missing values, and inconsistencies can compromise the validity of analytical results. Moreover, clean data facilitates more effective modeling and pattern recognition, as algorithms perform optimally when fed high-quality, error-free input.

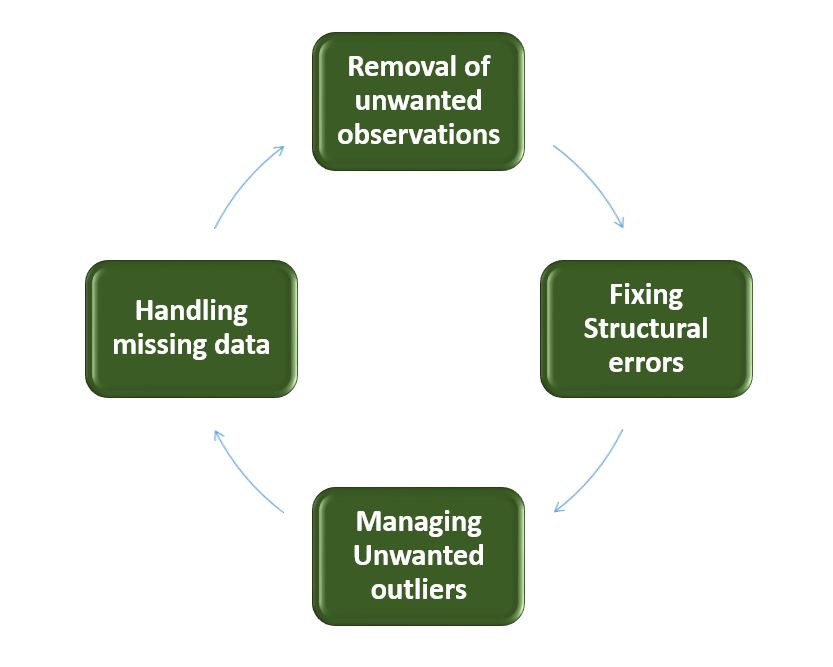
Additionally, clean datasets enhance the interpretability of findings, aiding in the formulation of actionable insights.

**Data Cleaning in Data Science**

Data clean-up is an integral component of data science, playing a fundamental role in ensuring the accuracy and reliability of datasets. In the field of data science, where insights and predictions are drawn from vast and complex datasets, the quality of the input data significantly influences the validity of analytical results. Data cleaning involves the systematic identification and correction of errors, inconsistencies, and inaccuracies within a dataset, encompassing tasks such as handling missing values, removing duplicates, and addressing outliers. This meticulous process is essential for enhancing the integrity of analyses, promoting more accurate modeling, and ultimately facilitating informed decision-making based on trustworthy and high-quality data.

**Steps to Perform Data Cleanliness**

Performing data cleaning involves a systematic process to identify and rectify errors, inconsistencies, and inaccuracies in a dataset. The following are essential steps to perform data cleaning.



*Data Cleaning*

* **Removal of Unwanted Observations**: Identify and eliminate irrelevant or redundant observations from the dataset. The step involves scrutinizing data entries for duplicate records, irrelevant information, or data points that do not contribute meaningfully to the analysis. Removing unwanted observations streamlines the dataset, reducing noise and improving the overall quality.
* **Fixing Structure errors:**Address structural issues in the dataset, such as inconsistencies in data formats, naming conventions, or variable types. Standardize formats, correct naming discrepancies, and ensure uniformity in [data representation](https://www.geeksforgeeks.org/what-are-the-different-ways-of-data-representation/). Fixing structure errors enhances data consistency and facilitates accurate analysis and interpretation.
* **Managing Unwanted outliers:**Identify and manage outliers, which are data points significantly deviating from the norm. Depending on the context, decide whether to remove outliers or transform them to minimize their impact on analysis. Managing outliers is crucial for obtaining more accurate and reliable insights from the data.
* **Handling Missing Data:** Devise strategies to handle missing data effectively. This may involve imputing missing values based on statistical methods, removing records with missing values, or employing advanced imputation techniques. Handling missing data ensures a more complete dataset, preventing biases and maintaining the integrity of analyses.

**How to Perform Data Cleanliness**

Performing data cleansing involves a systematic approach to enhance the quality and reliability of a dataset. The process begins with a thorough understanding of the data, inspecting its structure and identifying issues such as missing values, duplicates, and outliers. Addressing missing data involves strategic decisions on imputation or removal, while duplicates are systematically eliminated to reduce redundancy. Managing outliers ensures that extreme values do not unduly influence analysis. Structural errors are corrected to standardize formats and variable types, promoting consistency.

Throughout the process, documentation of changes is crucial for transparency and reproducibility. Iterative validation and testing confirm the effectiveness of the data cleansing steps, ultimately resulting in a refined dataset ready for meaningful analysis and insights.

**Python Implementation for Database Cleaning**

Let’s understand each step for Database Cleaning, using titanic dataset. Below are the necessary steps:

* Import the necessary libraries
* Load the dataset
* Check the data information using df.info()

Python

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

*# Load the dataset*

df = pd.read\_csv('titanic.csv')

df.head()

**Output**:

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

**Data Inspection and Exploration**

Let’s first understand the data by inspecting its structure and identifying missing values, outliers, and inconsistencies and check the duplicate rows with below python code:

Python

df.duplicated()

**Output**:

0 False  
1 False  
2 False  
3 False  
4 False  
 ...   
886 False  
887 False  
888 False  
889 False  
890 False  
Length: 891, dtype: bool

**Check the data information using df.info()**

Python

df.info()

**Output**:

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 PassengerId 891 non-null int64   
 1 Survived 891 non-null int64   
 2 Pclass 891 non-null int64   
 3 Name 891 non-null object   
 4 Sex 891 non-null object   
 5 Age 714 non-null float64  
 6 SibSp 891 non-null int64   
 7 Parch 891 non-null int64   
 8 Ticket 891 non-null object   
 9 Fare 891 non-null float64  
 10 Cabin 204 non-null object   
 11 Embarked 889 non-null object   
dtypes: float64(2), int64(5), object(5)  
memory usage: 83.7+ KB

From the above data info, we can see that Age and Cabin have an **unequal number of counts**. And some of the columns are categorical and have data type objects and some are integer and float values.

**Check the Categorical and Numerical Columns.**

Python

*# Categorical columns*

cat\_col = [col **for** col **in** df.columns **if** df[col].dtype == 'object']

print('Categorical columns :',cat\_col)

*# Numerical columns*

num\_col = [col **for** col **in** df.columns **if** df[col].dtype != 'object']

print('Numerical columns :',num\_col)

**Output**:

Categorical columns : ['Name', 'Sex', 'Ticket', 'Cabin', 'Embarked']  
Numerical columns : ['PassengerId', 'Survived', 'Pclass', 'Age', 'SibSp', 'Parch', 'Fare']

**Check the total number of Unique Values in the Categorical Columns**

Python

df[cat\_col].nunique()

**Output**:

Name 891  
Sex 2  
Ticket 681  
Cabin 147  
Embarked 3  
dtype: int64

Steps to Perform Data Cleansing

**Removal of all Above Unwanted Observations**

This includes deleting duplicate/ redundant or irrelevant values from your dataset. Duplicate observations most frequently arise during data collection and Irrelevant observations are those that don’t actually fit the specific problem that you’re trying to solve.

* Redundant observations alter the efficiency to a great extent as the data repeats and may add towards the correct side or towards the incorrect side, thereby producing unfaithful results.
* Irrelevant observations are any type of data that is of no use to us and can be removed directly.

**Now we have to make a decision according to the subject of analysis, which factor is important for our discussion.**

As we know our machines don’t understand the text data. So, we have to either drop or convert the categorical column values into numerical types. Here we are dropping the Name columns because the Name will be always unique and it hasn’t a great influence on target variables. For the ticket, Let’s first print the 50 unique tickets.

Python

df['Ticket'].unique()[:50]

**Output**:

array(['A/5 21171', 'PC 17599', 'STON/O2. 3101282', '113803', '373450',  
 '330877', '17463', '349909', '347742', '237736', 'PP 9549',  
 '113783', 'A/5. 2151', '347082', '350406', '248706', '382652',  
 '244373', '345763', '2649', '239865', '248698', '330923', '113788',  
 '347077', '2631', '19950', '330959', '349216', 'PC 17601',  
 'PC 17569', '335677', 'C.A. 24579', 'PC 17604', '113789', '2677',  
 'A./5. 2152', '345764', '2651', '7546', '11668', '349253',  
 'SC/Paris 2123', '330958', 'S.C./A.4. 23567', '370371', '14311',  
 '2662', '349237', '3101295'], dtype=object)

From the above tickets, we can observe that it is made of two like first values ‘A/5 21171’ is joint from of ‘A/5’ and  ‘21171’ this may influence our target variables. It will the case of **Feature Engineering**. where we derived new features from a column or a group of columns. In the current case, we are dropping the “Name” and “Ticket” columns.

**Drop Name and Ticket Columns**

Python

df1 = df.drop(columns=['Name','Ticket'])

df1.shape

**Output**:

(891, 10)

**Handling Missing Data**

Missing data is a common issue in real-world datasets, and it can occur due to various reasons such as human errors, system failures, or data collection issues. Various techniques can be used to handle missing data, such as imputation, deletion, or substitution.

Let’s check the % missing values columns-wise for each row using df.isnull() it checks whether the values are null or not and gives returns boolean values. and .sum() will sum the total number of null values rows and we divide it by the total number of rows present in the dataset then we multiply to get values in % i.e per 100 values how much values are null.

Python

round((df1.isnull().sum()/df1.shape[0])\*100,2)

**Output**:

PassengerId 0.00  
Survived 0.00  
Pclass 0.00  
Sex 0.00  
Age 19.87  
SibSp 0.00  
Parch 0.00  
Fare 0.00  
Cabin 77.10  
Embarked 0.22  
dtype: float64

We cannot just ignore or remove the missing observation. They must be handled carefully as they can be an indication of something important.

**The two most common ways to deal with missing data are:**

* **Dropping Observations with missing values**.
  + The fact that the value was missing may be informative in itself.
  + Plus, in the real world, you often need to make predictions on new data even if some of the features are missing!

As we can see from the above result that Cabin has 77% null values and Age has 19.87% and Embarked has 0.22% of null values.

So, it’s not a good idea to fill 77% of null values. So, we will drop the Cabin column. Embarked column has only 0.22% of null values so, we drop the null values rows of Embarked column.

Python

df2 = df1.drop(columns='Cabin')

df2.dropna(subset=['Embarked'], axis=0, inplace=**True**)

df2.shape

**Output**:

(889, 9)

* **Imputing the missing values from past observations.**
  + Again, “missingness” is almost always informative in itself, and you should tell your algorithm if a value was missing.
  + Even if you build a model to impute your values, you’re not adding any real information. You’re just reinforcing the patterns already provided by other features.

We can use **Mean imputation** or **Median imputations** for the case.

**Note:**

* Mean imputation is suitable when the data is normally distributed and has no extreme outliers.
* Median imputation is preferable when the data contains outliers or is skewed.

Python

*# Mean imputation*

df3 = df2.fillna(df2.Age.mean())

*# Let's check the null values again*

df3.isnull().sum()

**Output**:

PassengerId 0  
Survived 0  
Pclass 0  
Sex 0  
Age 0  
SibSp 0  
Parch 0  
Fare 0  
Embarked 0  
dtype: int64

**Handling Outliers**

Outliers are extreme values that deviate significantly from the majority of the data. They can negatively impact the analysis and model performance. Techniques such as clustering, interpolation, or transformation can be used to handle outliers.

To check the outliers, We generally use a box plot. A box plot, also referred to as a box-and-whisker plot, is a graphical representation of a dataset’s distribution. It shows a variable’s median, quartiles, and potential outliers. The line inside the box denotes the median, while the box itself denotes the interquartile range (IQR). The whiskers extend to the most extreme non-outlier values within 1.5 times the IQR. Individual points beyond the whiskers are considered potential outliers. A box plot offers an easy-to-understand overview of the range of the data and makes it possible to identify outliers or skewness in the distribution.

**Let’s plot the box plot for Age column data.**

Python

**import** **matplotlib.pyplot** **as** **plt**

plt.boxplot(df3['Age'], vert=**False**)

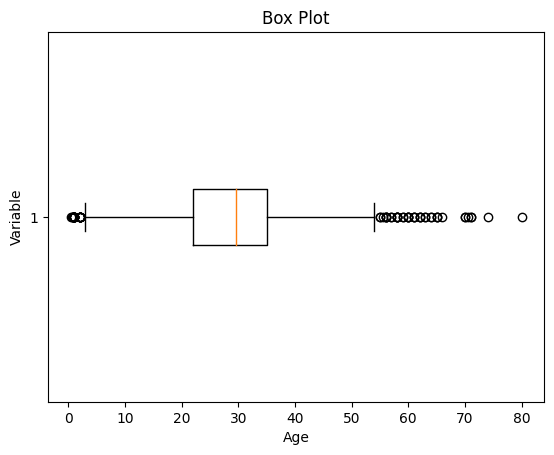
plt.ylabel('Variable')

plt.xlabel('Age')

plt.title('Box Plot')

plt.show()

**Output**:



*Box Plot*

As we can see from the above Box and whisker plot, Our age dataset has outliers values. The values less than 5 and more than 55 are outliers.

Python

*# calculate summary statistics*

mean = df3['Age'].mean()

std = df3['Age'].std()

*# Calculate the lower and upper bounds*

lower\_bound = mean - std\*2

upper\_bound = mean + std\*2

print('Lower Bound :',lower\_bound)

print('Upper Bound :',upper\_bound)

*# Drop the outliers*

df4 = df3[(df3['Age'] >= lower\_bound)

& (df3['Age'] <= upper\_bound)]

**Output**:

Lower Bound : 3.705400107925648  
Upper Bound : 55.578785285332785

Similarly, we can remove the outliers of the remaining columns.

**Data Transformation**

Data transformation involves converting the data from one form to another to make it more suitable for analysis. Techniques such as normalization, scaling, or encoding can be used to transform the data.

**Data validation and verification**

Data validation and verification involve ensuring that the data is accurate and consistent by comparing it with external sources or expert knowledge.

For the machine learning prediction, First, we separate independent and target features. Here we will consider only **‘Sex’ ‘Age’ ‘SibSp’, ‘Parch’ ‘Fare’ ‘Embarked’**only as the independent features and **Survived**as target variables. Because PassengerId will not affect the survival rate.

Python

X = df3[['Pclass','Sex','Age', 'SibSp','Parch','Fare','Embarked']]

Y = df3['Survived']

**Data formatting**

Data formatting involves converting the data into a standard format or structure that can be easily processed by the algorithms or models used for analysis. Here we will discuss commonly used data formatting techniques i.e. Scaling and Normalization.

**Scaling**

* Scaling involves transforming the values of features to a specific range. It maintains the shape of the original distribution while changing the scale.
* Particularly useful when features have different scales, and certain algorithms are sensitive to the magnitude of the features.
* Common scaling methods include Min-Max scaling and Standardization (Z-score scaling).

**Min-Max Scaling**: Min-Max scaling rescales the values to a specified range, typically between 0 and 1. It preserves the original distribution and ensures that the minimum value maps to 0 and the maximum value maps to 1.

Python

**from** **sklearn.preprocessing** **import** MinMaxScaler

*# initialising the MinMaxScaler*

scaler = MinMaxScaler(feature\_range=(0, 1))

*# Numerical columns*

num\_col\_ = [col **for** col **in** X.columns **if** X[col].dtype != 'object']

x1 = X

*# learning the statistical parameters for each of the data and transforming*

x1[num\_col\_] = scaler.fit\_transform(x1[num\_col\_])

x1.head()

**Output**:

Pclass Sex Age SibSp Parch Fare Embarked  
0 1.0 male 0.271174 0.125 0.0 0.014151 S  
1 0.0 female 0.472229 0.125 0.0 0.139136 C  
2 1.0 female 0.321438 0.000 0.0 0.015469 S  
3 0.0 female 0.434531 0.125 0.0 0.103644 S  
4 1.0 male 0.434531 0.000 0.0 0.015713 S

**Standardization (Z-score scaling):**Standardization transforms the values to have a mean of 0 and a standard deviation of 1. It centers the data around the mean and scales it based on the standard deviation. Standardization makes the data more suitable for algorithms that assume a Gaussian distribution or require features to have zero mean and unit variance.

Z = (X - μ) / σ

Where,

* X = Data
* μ = Mean value of X
* σ = Standard deviation of X

**Data Cleansing Tools**

Some data cleansing tools**:**

* OpenRefine
* Trifacta Wrangler
* TIBCO Clarity
* Cloudingo
* IBM Infosphere Quality Stage

**Advantages of Data Cleaning in Machine Learning:**

* **Improved model performance**: Removal of errors, inconsistencies, and irrelevant data, helps the model to better learn from the data.
* **Increased accuracy**: Helps ensure that the data is accurate, consistent, and free of errors.
* Better representation of the data: Data cleaning allows the data to be transformed into a format that better represents the underlying relationships and patterns in the data.
* Improved data quality: Improve the quality of the data, making it more reliable and accurate.
* Improved data security: Helps to identify and remove sensitive or confidential information that could compromise data security.

**Disadvantages of Data Cleaning in Machine Learning**

* Time-consuming: Time-Consuming task, especially for large and complex datasets.
* Error-prone: Data cleaning can be error-prone, as it involves transforming and cleaning the data, which can result in the loss of important information or the introduction of new errors.
* Cost and resource-intensive: Resource-intensive process that requires significant time, effort, and expertise. It can also require the use of specialized software tools, which can add to the cost and complexity of data cleaning.
* Overfitting: Data cleaning can inadvertently contribute to overfitting by removing too much data.

**Handling duplicate records:**

Finding duplicates in the dataset is the first step in addressing them. A number of functions are available in the pandas library to find duplicates. If a row is a duplicate of another row, the duplicated method returns a Boolean Series that says so. Duplicate rows are removed from a dataset using the drop duplicates function.

An illustration of how to spot duplicate values in a pandas DataFrame is given below −

Example

Open Compiler

import pandas as pd

# Create a sample DataFrame with duplicate values

data = pd.DataFrame({

'name': ['John', 'Emily', 'John', 'Jane', 'John'],

'age': [25, 28, 25, 30, 25],

'salary': [50000, 60000, 50000, 70000, 50000]

})

# Identify duplicate rows

duplicates = data.duplicated()

# Print the duplicate rows

print(data[duplicates])

Output

name age salary

2 John 25 50000

4 John 25 50000

Duplicate values in a Pandas DataFrame may be found and printed using the provided Python code. The code is broken down as follows −

* The Pandas library is initially imported as pd.
* There are duplicate entries in the three columns for name, age, and income in a sample DataFrame.
* To find duplicate rows in the DataFrame, utilize the Pandas duplicated() function. For each row that is a duplicate of a prior row, the procedure produces a Boolean Series that contains the value True.
* Square brackets are used to index the original DataFrame in the Boolean Series. Only the duplicate rows are returned in this case.
* The final step is to print the DataFrame with duplicate rows to the console.

A DataFrame comprising the rows that are duplicates of earlier rows based on all the columns will be the result of this code.

**Feature scaling, normalization, and encoding.**

**Why use Feature Scaling?**

In machine learning, feature scaling is employed for a number of purposes:

* Scaling guarantees that all features are on a comparable scale and have comparable ranges. This process is known as feature normalisation. This is significant because the magnitude of the features has an impact on many machine learning techniques. Larger scale features may dominate the learning process and have an excessive impact on the outcomes. You can avoid this problem and make sure that each feature contributes equally to the learning process by scaling the features.
* Algorithm performance improvement: When the features are scaled, several machine learning methods, including gradient descent-based algorithms, distance-based algorithms (such k-nearest neighbours), and support vector machines, perform better or converge more quickly. The algorithm’s performance can be enhanced by scaling the features, which can hasten the convergence of the algorithm to the ideal outcome.
* Preventing numerical instability: Numerical instability can be prevented by avoiding significant scale disparities between features. Examples include distance calculations or matrix operations, where having features with radically differing scales can result in numerical overflow or underflow problems. Stable computations are ensured and these issues are mitigated by scaling the features.
* Scaling features makes ensuring that each characteristic is given the same consideration during the learning process. Without scaling, bigger scale features could dominate the learning, producing skewed outcomes. This bias is removed through scaling, which also guarantees that each feature contributes fairly to model predictions.

**Absolute Maximum Scaling**

This method of scaling requires two-step:

1. We should first select the maximum absolute value out of all the entries of a particular measure.
2. Then after this, we divide each entry of the column by this maximum value.

After performing the above-mentioned two steps we will observe that each entry of the column lies in the range of -1 to 1. But this method is not used that often the reason behind this is that it is too sensitive to the outliers. And while dealing with the real-world data presence of outliers is a very common thing.

For the demonstration purpose, we will use the dataset which you can download from [here](https://drive.google.com/file/d/1J7dPhnj2yBuzPwYraFU6cpCsa8Va3fiM/view?usp=share_link). This dataset is a simpler version of the original house price prediction dataset having only two columns from the original dataset. The first five rows of the original data are shown below:

* Python3

|  |
| --- |
| **import** pandas as pd  df **=** pd.read\_csv('SampleFile.csv')  print(df.head()) |

**Output:**

LotArea MSSubClass  
0 8450 60  
1 9600 20  
2 11250 60  
3 9550 70  
4 14260 60

Now let’s apply the first method which is of the absolute maximum scaling. For this first, we are supposed to evaluate the absolute maximum values of the columns.

* Python3

|  |
| --- |
| **import** numpy as np  max\_vals **=** np.max(np.abs(df))  max\_vals |

**Output:**

LotArea 215245  
MSSubClass 190  
dtype: int64

Now we are supposed to subtract these values from the data and then divide the results from the maximum values as well.

* Python3

|  |
| --- |
| print((df **-** max\_vals) **/** max\_vals) |

**Output:**

LotArea MSSubClass  
0 -0.960742 -0.684211  
1 -0.955400 -0.894737  
2 -0.947734 -0.684211  
3 -0.955632 -0.631579  
4 -0.933750 -0.684211  
... ... ...  
1455 -0.963219 -0.684211  
1456 -0.938791 -0.894737  
1457 -0.957992 -0.631579  
1458 -0.954856 -0.894737  
1459 -0.953834 -0.894737  
  
[1460 rows x 2 columns]

**Min-Max Scaling**

This method of scaling requires below two-step:

1. First, we are supposed to find the minimum and the maximum value of the column.
2. Then we will subtract the minimum value from the entry and divide the result by the difference between the maximum and the minimum value.

As we are using the maximum and the minimum value this method is also prone to [outliers](https://www.geeksforgeeks.org/machine-learning-outlier/) but the range in which the data will range after performing the above two steps is between 0 to 1.

* Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** MinMaxScaler    scaler **=** MinMaxScaler()  scaled\_data **=** scaler.fit\_transform(df)  scaled\_df **=** pd.DataFrame(scaled\_data,                           columns**=**df.columns)  scaled\_df.head() |

**Output:**

LotArea MSSubClass  
0 0.033420 0.235294  
1 0.038795 0.000000  
2 0.046507 0.235294  
3 0.038561 0.294118  
4 0.060576 0.235294

**Normalization**

This method is more or less the same as the previous method but here instead of the minimum value, we subtract each entry by the mean value of the whole data and then divide the results by the difference between the minimum and the maximum value.

* Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** Normalizer    scaler **=** Normalizer()  scaled\_data **=** scaler.fit\_transform(df)  scaled\_df **=** pd.DataFrame(scaled\_data,                           columns**=**df.columns)  print(scaled\_df.head()) |

**Output:**

LotArea MSSubClass  
0 0.999975 0.007100  
1 0.999998 0.002083  
2 0.999986 0.005333  
3 0.999973 0.007330  
4 0.999991 0.004208

**Standardization**

This method of scaling is basically based on the central tendencies and variance of the data.

1. First, we should calculate the [mean and standard deviation](https://www.geeksforgeeks.org/mathematics-mean-variance-and-standard-deviation/) of the data we would like to normalize.
2. Then we are supposed to subtract the mean value from each entry and then divide the result by the standard deviation.

This helps us achieve a [normal distribution](https://www.geeksforgeeks.org/mathematics-probability-distributions-set-3-normal-distribution/)(if it is already normal but skewed) of the data with a mean equal to zero and a standard deviation equal to 1.

* Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** StandardScaler    scaler **=** StandardScaler()  scaled\_data **=** scaler.fit\_transform(df)  scaled\_df **=** pd.DataFrame(scaled\_data,                           columns**=**df.columns)  print(scaled\_df.head()) |

**Output:**

LotArea MSSubClass  
0 -0.207142 0.073375  
1 -0.091886 -0.872563  
2 0.073480 0.073375  
3 -0.096897 0.309859  
4 0.375148 0.073375

***Thank you***