Pandas For

Data Cleaning

What is Pandas?

Pandas is a python library used for working with dataset, and it is open-source library.

It has functions for analyzing, cleaning, exploring and manipulating data.

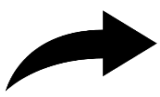
Pandas was created by Wes McKinney in 2008.

It also integrates seamlessly with other popular python libraries. such as NumPy for numerical computing and matplotlib for data visualization. this make it is a powerful asset for data driven tasks.

Why Use Pandas?

It allows us to analyze bigdata and make conclusion based on statistical thories.

Pandas can clean messy data and make them readable and relavant.



Installation of pandas:

Pip install pandas

```
In [1]: !pip install pandas
```

How to Import the Necessary Libraries?

To save time and typing, we often import Pandas as pd.

This lets us use the shorter pd.read_csv() instead of pandas.read_csv() for reading CSV files, making our code more efficient and readable.

```
In [2]: import pandas as pd
        import numpy as np
```


Pandas has two primary data structures.

1.What is Series?

It is a one-dimensional array holding data of any type.

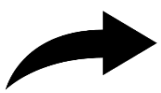
A Pandas Series is like a column in a table.

```
In [8]: a = [1,2,3]
        data =pd.Series(a)
        print(data)

        0    1
        1    2
        2    3
        dtype: int64
```

Labels

Lables are nothing else is specified the s values are labelled with their index number.

```
In [7]: a = [1,2,3]
        data =pd.Series(a)
        print(data[1])

        2
```

Create Labels

With index argument u can name your own lables.

```
In [6]: #create lable
        a = [1,2,3]
        data =pd.Series(a,index=["a","b","c"])
        print(data)
```

```
a    1
b    2
c    3
dtype: int64
```

2.DataFrame

A Pandas DataFrame is a 2 dimensional data structure, like 2-D array, or a table with rows and columns.

e.g:

```
In [40]: data = { "id" : [1,2,3],
                  "marks":[89,98,76]
                }
         df= pd.DataFrame(data,index=["Sunil","krish","Piyush"])
         print(df)
```

```
         id  marks
Sunil     1     89
krish     2     98
Piyush    3     76
```

Locate Named Indexes

Use the named index in the loc attribute to return the specified row(s).

e.g:

```
In [42]: print(df.loc["Piyush"])

id         3
marks     76
Name: Piyush, dtype: int64
```

Rename the column name

```
In [51]: a.rename(columns = {"name" : "first_name"}, inplace =  True )   #inplace is used for permanent changes
         a
```

Out[51]:

	first_name	last_name	City
0	Atharava	konde	Pune
1	Rahul	patil	Dharashiv
2	Shashi	Bhosale	Pune
3	Mansi	Taur	Dharashiv

Swap the rows name to column using transpose function

```
In [62]: d1
```

Out[62]:

	C1	C2	C3
f_name	Akshad	Akshay	Ajay
l_name	Patil	Sharma	Sawant
Roll_no	A26	A27	A28
Marks	34	56	78

```
In [63]: d1= d1.transpose() #transpose() method is used to swap the rows and columns of a DataFrame
         d1
```

Out[63]:

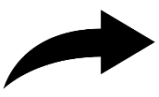
	f_name	l_name	Roll_no	Marks
C1	Akshad	Patil	A26	34
C2	Akshay	Sharma	A27	56
C3	Ajay	Sawant	A28	78

Give column and rows name simultaneously

```
In [59]: lst1 = ["Akshad","Akshay","Ajay"]          # give column and row name at time
         lst2 = ["Patil","Sharma","Sawant"]
         lst3 = ["A26","A27","A28"]
         lst4 = [34,56,78]

         d1 = pd.DataFrame([lst1,lst2,lst3,lst4],columns = ["C1","C2","C3"],index= ["f_name","l_name","Roll_no","Marks"])
         d1
```

Out[59]:

	C1	C2	C3
f_name	Akshad	Akshay	Ajay
l_name	Patil	Sharma	Sawant
Roll_no	A26	A27	A28
Marks	34	56	78

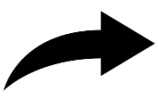


Swap the column name to the rows using dictionary & list

In [58]:
```
dict2 = {
    "science" : [45,78,97,96,75],
   " history ": [49,97,85,87,87],
    "maths " : [55,64,67,77,87]
}
#change rows name
name = ["Ajay","Vijay","Gayatri","Krish","Karina"]
df5 = pd.DataFrame(dict2, index = name)
df5
```

Out[58]:

	science	history	maths
Ajay	45	49	55
Vijay	78	97	64
Gayatri	97	85	67
Krish	96	87	77
Karina	75	87	87



What is data cleaning?

Before we dive into our data journey with Pandas, let's take a moment to demystify "data cleaning." Think of it as giving your dataset a fresh start, ensuring everything is in its place and spotless for accurate analysis.

Data cleaning involves spotting and fixing errors, inconsistencies, and missing values in your dataset. It's like organizing your tools before starting a project; you need everything in order for the best results.

Why is data cleaning important? Imagine trying to uncover customer insights with missing information, or making decisions based on data full of duplicates. It would be frustrating and unreliable.

With data cleaning, using tools like Pandas, we eliminate inconsistencies, correct errors, and reveal the true potential of your data, ensuring your analysis is both precise and insightful.

What is Data Processing?

You might be wondering, "Do data cleaning and data preprocessing mean the same thing?" The answer is no – they serve different roles in data preparation.

Imagine you have a raw piece of marble. Data cleaning is like chiseling away the rough edges, removing any impurities, and revealing the stone's true quality.

On the other hand, data preprocessing is akin to sculpting that piece of marble into a refined statue, preparing it for display. It goes beyond cleaning; it involves transforming and optimizing the data for specific analyses or tasks.

Data cleaning is the initial step, making your dataset readable and usable by removing duplicates, handling missing values, and converting data types.

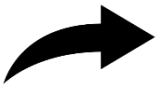
Data preprocessing takes this clean data further, applying advanced techniques like feature engineering, encoding categorical variables, and managing outliers to improve and refine the data for more sophisticated analyses or machine learning models.

The ultimate goal is to transform your dataset into a polished work of art, ready for in-depth analysis or modeling.

How to Load the Dataset?

Start by loading my_dataset into a Pandas DataFrame. In this example, we'll use a hypothetical dataset named my_dataset.csv. We will load the dataset into a variable called df.

```
In [ ]:
#Replace 'your_dataset.csv' with the actual dataset name or file path
df = pd.read_csv('my_dataset.csv')
```

Exploratory Data Analysis (EDA)?

EDA helps you understand the structure and characteristics of your dataset. Some Pandas functions help us gain insights into our dataset. We call these functions by calling the dataset variable plus the function.

For example:

df.head() will call the first 5 rows of the dataset. You can specify the number of rows to be displayed in the parentheses.

df.describe() gives some statistical data like percentile, mean and standard deviation of the numerical values of the Series or DataFrame.

df.info() gives the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column(non-null values).

```
In [ ]:  #Display the first few rows of the dataset
         print(df.head())

         #Summary statistics
         print(df.describe())

         #Information about the dataset
         print(df.info())
```

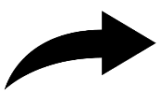
Handling Missing Values?

As someone new to this field, dealing with missing values can be quite daunting because they appear in various forms and can significantly affect your analysis or model performance.

Machine learning models require complete data, and missing or "NaN" values can distort your final results. However, don't worry—Pandas offers several methods to tackle this issue effectively.

One way to do this is by removing the missing values altogether. Code snippet below :

```
In [ ]:
#Check for missing values
print(df.isnull().sum())

#Drop rows with missing values and place it in a new variable "df_cleaned"
df_cleaned = df.dropna()

#Fill missing values with mean for numerical data and place it ina new variable called df_filled
df_filled = df.fillna(df.mean())
```

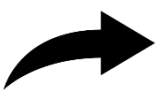
But If the number of rows with missing values is large, simply removing them might not be adequate.

Instead, for numerical data, you can compute the mean and input it into the rows with missing values. Here's a code snippet to demonstrate this:

```
In [ ]:
#Replace missing values with the mean of each column
df.fillna(df.mean(), inplace=True)

#If you want to replace missing values in a specific column, you can do it this way:
#Replace 'column_name' with the actual column name
df['column_name'].fillna(df['column_name'].mean(), inplace=True)


#Now, df contains no missing values, and NaNs have been replaced with column mean
```

How to Remove Duplicate Records?

Duplicate records can distort your analysis by influencing the results in ways that do not accurately show trends and underlying patterns (by producing outliers). Pandas helps to identify and remove the duplicate values in an easy way by placing them in new variables.



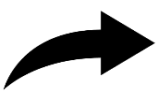
```
In [ ]:  #Identify duplicates
         print(df.duplicated().sum())

         #Remove duplicates
         df_no_duplicates = df.drop_duplicates()
```

Data Types and Conversion

Data type conversion in Pandas is a crucial aspect of data preprocessing, allowing you to ensure that your data is in the appropriate format for analysis or modeling.

Data from various sources are usually messy and the data types of some values may be in the wrong format, for example some numerical values may come in 'float' or 'string' format instead of 'integer' format and a mix up of these formats leads to errors and wrong results.

You can convert a Column of type int to float with the following code:



```
In [ ]:  #Convert 'Column1' to float
         df['Column1'] = df['Column1'].astype(float)

         #Display updated data types
         print(df.dtypes)
```

You can use df.dtypes to print column data types.

How to Handle Outliers

Outliers are data points significantly different from the majority of the data, they can distort statistical measures and adversely affect the performance of machine learning models.

They may be caused by human error, missing NaN values, or could be accurate data that does not correlate with the rest of the data.

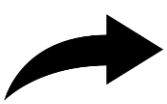
There are several methods to identify and remove outliers, they are:

- Remove NaN values.
- Visualize the data before and after removal.
- Z-score method (for normally distributed data).
- IQR (Interquartile range) method for more robust data.

The IQR is useful for identifying outliers in a dataset. According to the IQR method, values that fall below Q1−1.5× IQR or above Q3+1.5×IQR are considered outliers.

This rule is based on the assumption that most of the data in a normal distribution should fall within this range. Here's a code snippet for the IQR method:

In [ ]:
```
#Using median calculations and IQR, outliers are identified and these data points should be removed
Q1 = df["column_name"].quantile(0.25)
Q3 = df["column_name"].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df = df[df["column_name"].between(lower_bound, upper_bound)]
```

Thank You

@VaishnaviDauale