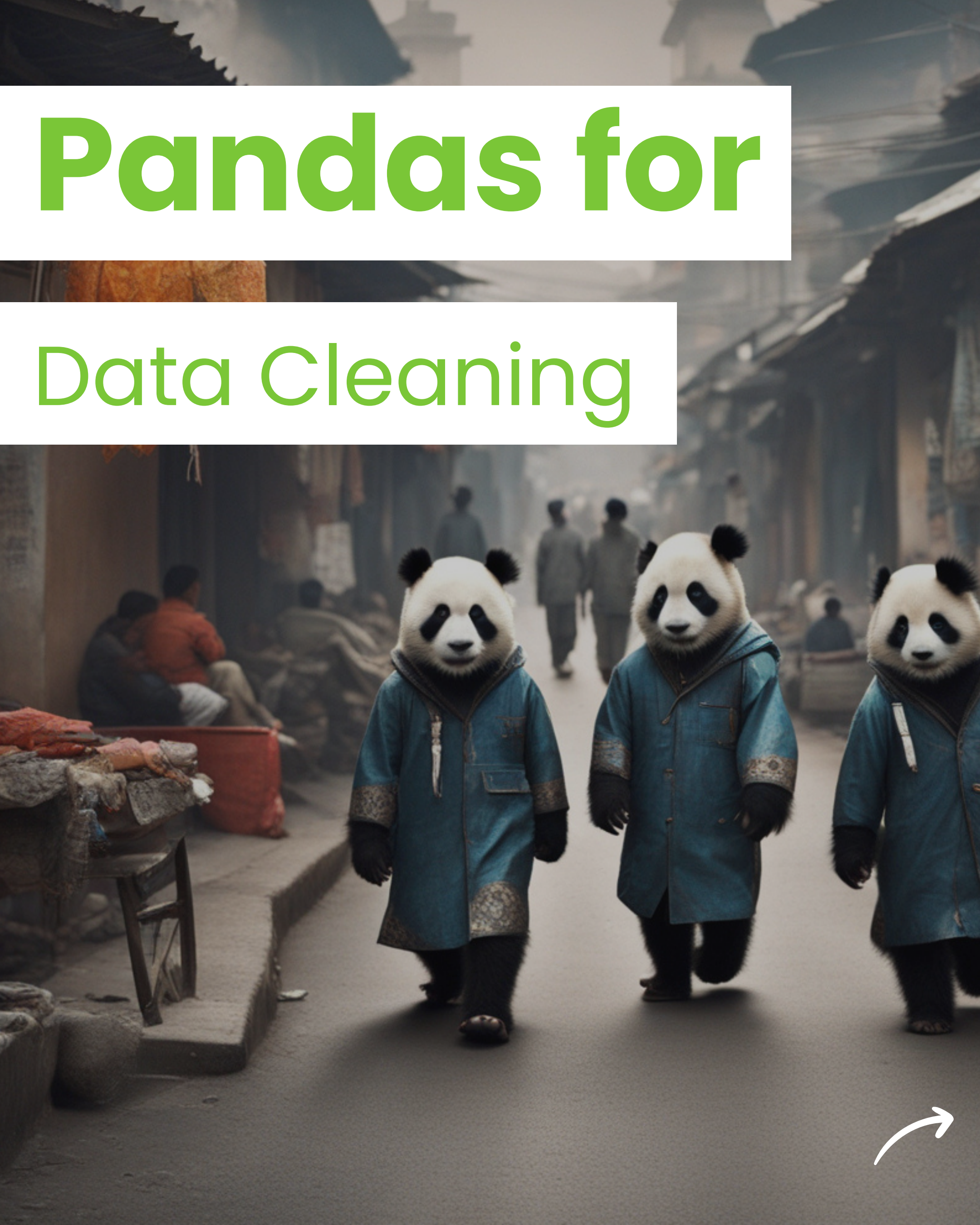
Pandas for

Data Cleaning



What is Pandas?

Pandas is a popular open-source data manipulation and analysis library for Python.

It provides easy-to-use functions needed to work with structured data seamlessly.

Pandas also integrates seamlessly with other popular Python libraries, such as NumPy for numerical computing and Matplotlib for data visualization. This makes it a powerful asset for data driven tasks.

Pandas excels in handling missing data, reshaping datasets, merging and joining multiple datasets, and performing complex operations on data, making it exceptionally useful for data cleaning and manipulation.

What is Data Cleaning?

Before we embark on our data adventure with Pandas, let's take a moment to explain the term "data cleaning." Think of it as the digital detox for your dataset, where we tidy up, and and prioritize accuracy above all else.

Data cleaning involves identifying and rectifying errors, inconsistencies, and missing values within a dataset. It's like preparing your ingredients before cooking; you want everything in order to get the perfect analysis or visualization.

Why bother with data cleaning? Well, imagine trying to analyze sales trends when some entries are missing, or working with a dataset that has duplicate records throwing off your calculations.

Not ideal, right?

In this digital detox, we use tools like Pandas to get rid of inconsistencies, straighten out errors, and let the true clarity of your data shine through.

What is Data Processing?

You may be wondering, "Does data cleaning and data preprocessing mean the same thing?" The answer is no – they do not.

Picture this: you stumble upon an ancient treasure chest buried in the digital sands of your dataset. Data cleaning is like carefully unearthing that chest, dusting off the cobwebs, and ensuring that what's inside is authentic and reliable.

As for data preprocessing, you can think of it as taking that discovered treasure and preparing its contents for public display. It goes beyond cleaning; it's about transforming and optimizing the data for specific analyses or tasks.

Data cleaning is the initial phase of refining your dataset, making it readable and usable with techniques like removing duplicates, handling missing values and data type conversion.

Data preprocessing is similar to taking this refined data and scaling with more advanced techniques such as feature engineering, encoding categorical variables and and handling outliers to achieve better and more advanced results.

The goal is to turn your dataset into a refined masterpiece, ready for analysis or modeling.

How to Import the Necessary Libraries

Before we embark on data cleaning and preprocessing, let's import the Pandas library.

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.

```
import pandas as pd
```

How to Load the Dataset

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

```
#Replace 'your_dataset.csv' with the actual dataset name or file path

df = pd.read_csv('your_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).

```
#Display the first few rows of the dataset

print(df.head())

#Summary statistics

print(df.describe())

#Information about the dataset

print(df.info())
```

How to Handle Missing Values

As a newbie in this field, missing values pose a significant stress as they come in different formats and can adversely impact your analysis or model.

Machine learning models cannot be trained with data that has missing or "NAN" values as they can alter your end result during analysis. But do not fret, Pandas provides methods to handle this problem.

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

```
#Check for missing values

print(df.isnull().sum())

#Drop rows with missing valiues 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 that have missing values is large, then this method will be inadequate.

For numerical data, you can simply compute the mean and input it into the rows that have missing values. Code snippet below:

```
#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.

Code snippet below:

```
#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:

```
#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:

```
#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)]
```


If you find this helpful, Repost

+ Follow for more content.

linkedin.com/in/ileonjose