**Introduction to Pandas**

Pandas is a powerful open-source data manipulation and analysis library for Python. It provides flexible data structures to handle structured and semi-structured data efficiently. Pandas is built on top of NumPy and integrates well with other Python libraries.

**Pandas Data Structures**

Pandas provides three primary data structures:

1. **Series**
   * A one-dimensional labeled array capable of holding any data type.
   * Created using pd.Series().
   * Example:
   * import pandas as pd
   * s = pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
   * print(s)
2. **DataFrame**
   * A two-dimensional labeled data structure similar to a table.
   * Created using pd.DataFrame().
   * Example:
   * data = {'Name': ['Alice', 'Bob', 'Charlie'],
   * 'Age': [25, 30, 35],
   * 'City': ['New York', 'Los Angeles', 'Chicago']}
   * df = pd.DataFrame(data)
   * print(df)
3. **Panel (Deprecated in Pandas 1.0.0)**
   * A three-dimensional data structure (used previously but now replaced by MultiIndex DataFrames).
   * Instead of Panel, MultiIndex DataFrames are used:
   * arrays = [['A', 'A', 'B', 'B'], ['one', 'two', 'one', 'two']]
   * index = pd.MultiIndex.from\_arrays(arrays, names=('Letter', 'Number'))
   * df = pd.DataFrame([[1, 2], [3, 4], [5, 6], [7, 8]], index=index, columns=['Value1', 'Value2'])
   * print(df)

**Handling Missing Data**

Missing data is a common issue in real-world datasets. Pandas provides methods to handle missing values.

1. **isnull()** – Checks for missing values:  
   df = pd.DataFrame({'A': [1, 2, None], 'B': [None, 3, 4]})  
   print(df.isnull())
2. **dropna()** – Removes missing values:  
   df.dropna() # Drops rows with NaN  
   df.dropna(axis=1) # Drops columns with NaN
3. **fillna()** – Replaces missing values:  
   df.fillna(value=0) # Replaces NaN with 0  
   df.fillna(method='ffill') # Forward fills NaN values

**Renaming Columns and Rows**

To rename columns or index labels:

df.rename(columns={'A': 'Column1', 'B': 'Column2'}, inplace=True)

df.rename(index={0: 'Row1', 1: 'Row2'}, inplace=True)

**Data Transformation**

Transformations help apply functions to modify data.

1. **apply()** – Applies a function along a DataFrame’s axis:  
   df['A'] = df['A'].apply(lambda x: x \* 2)
2. **map()** – Works on Series:  
   df['A'] = df['A'].map({1: 'One', 2: 'Two'})
3. **applymap()** – Applies a function element-wise:  
   df = df.applymap(lambda x: str(x) + '!')

**String Manipulations**

Pandas provides vectorized string operations via the .str accessor.

df['Name'] = df['Name'].str.upper()

df['Name'] = df['Name'].str.replace('Alice', 'Alicia')

**Grouping Data (groupby())**

Grouping allows aggregation on subsets of data.

df.groupby('City').sum()

**Data Aggregation**

Aggregation functions summarize data.

df.agg({'Age': ['sum', 'mean'], 'Salary': ['min', 'max']})

Common functions:

* sum(), mean(), count(), min(), max(), agg()

**Sorting and Ranking Data**

1. **Sorting**:  
   df.sort\_values(by='Age', ascending=False)  
   df.sort\_index()
2. **Ranking**:  
   df['Rank'] = df['Age'].rank()

**Merging, Joining, and Concatenating**

Pandas provides various ways to combine data.

1. **Concatenation (concat())**:  
   pd.concat([df1, df2])
2. **Merging (merge())**:  
   df1.merge(df2, on='ID', how='inner') # Types: inner, left, right, outer
3. **Joining (DataFrame.join())**:  
   df1.join(df2, lsuffix='\_left', rsuffix='\_right')
4. **Ordered Merge (merge\_ordered())** and **As-of Merge (merge\_asof())**:  
   pd.merge\_ordered(df1, df2, on='Date')  
   pd.merge\_asof(df1, df2, on='Timestamp')
5. **Comparing Data (compare())**:  
   df1.compare(df2)

**Reshaping and Pivot Tables**

1. **Pivot and Pivot Table**:  
   df.pivot(index='Date', columns='City', values='Sales')  
   df.pivot\_table(index='City', columns='Product', values='Sales', aggfunc='sum')
2. **Stack and Unstack**:  
   df.stack()  
   df.unstack()
3. **Melting Data**:  
   df.melt(id\_vars=['ID'], value\_vars=['Math', 'Science'])
4. **Dummy Variables (get\_dummies())**:  
   pd.get\_dummies(df, columns=['City'])
5. **Exploding Lists into Rows**:  
   df.explode('ColumnWithLists')
6. **Crosstab**:  
   pd.crosstab(df['City'], df['Gender'])
7. **Categorization (cut(), factorize())**:  
   pd.cut(df['Age'], bins=[0, 18, 35, 60], labels=['Teen', 'Young', 'Adult'])  
   df['Category'] = pd.factorize(df['City'])[0]

**Time Series Data in Pandas**

Pandas supports handling and analyzing time-series data.

df['Date'] = pd.to\_datetime(df['Date'])

df.set\_index('Date', inplace=True)

df.resample('M').sum() # Resampling by month

**Pandas for Visualization**

Pandas integrates with Matplotlib for plotting.

import matplotlib.pyplot as plt

df.plot(kind='line', x='Date', y='Sales')

plt.show()

For time-series data:

df['Sales'].plot()

plt.show()

**Conclusion**

Pandas is a powerful library for data manipulation, cleaning, transformation, and analysis. It provides robust functionality for handling missing data, reshaping data, merging datasets, time series analysis, and visualization, making it an essential tool for data scientists and analysts.

**Compute & Visualize Statistics in Pandas**

Pandas provides powerful tools for computing and visualizing statistics in datasets. This includes descriptive statistics, summary statistics, and data visualization using built-in plotting methods.

**1. Computing Statistics in Pandas**

Pandas offers a range of functions to compute statistics on datasets.

**Descriptive Statistics**

You can compute key statistical measures using the following methods:

import pandas as pd

import numpy as np

# Sample DataFrame

data = {

'Age': [25, 30, 35, 40, 50, np.nan, 60],

'Salary': [4000, 5000, 6000, 8000, 10000, np.nan, 12000]

}

df = pd.DataFrame(data)

# Summary statistics

print(df.describe()) # Provides count, mean, std, min, 25%, 50%, 75%, max

**Basic Statistics**

* **Mean (Average)**:
* df.mean()
* **Median (50th percentile)**:
* df.median()
* **Standard Deviation**:
* df.std()
* **Variance**:
* df.var()
* **Minimum and Maximum**:
* df.min()
* df.max()
* **Sum of Values**:
* df.sum()
* **Count of Non-NaN Values**:
* df.count()
* **Correlation Between Variables**:
* df.corr() # Pearson correlation coefficient
* **Covariance Between Variables**:
* df.cov()

**2. Visualizing Statistics in Pandas**

Pandas integrates with Matplotlib to provide quick and easy visualizations.

import matplotlib.pyplot as plt

import seaborn as sns

# Setting style for better visuals

sns.set(style="whitegrid")

**Histogram (Distribution of Data)**

A histogram is useful for understanding data distribution.

df['Age'].plot(kind='hist', bins=5, title="Age Distribution")

plt.xlabel("Age")

plt.show()

**Box Plot (Detecting Outliers)**

Box plots help visualize the spread and identify outliers.

df.boxplot(column=['Age', 'Salary'])

plt.title("Boxplot of Age & Salary")

plt.show()

**Scatter Plot (Correlation Between Two Variables)**

A scatter plot helps visualize relationships between variables.

df.plot(kind='scatter', x='Age', y='Salary', title="Age vs Salary")

plt.show()

**Line Plot (Trends Over Time)**

If data includes time series, a line plot can show trends.

df['Salary'].plot(kind='line', title="Salary Trends", marker='o')

plt.ylabel("Salary")

plt.show()

**Bar Chart (Comparing Categories)**

Bar charts compare categorical data.

df['Age'].value\_counts().plot(kind='bar', title="Age Count")

plt.xlabel("Age")

plt.ylabel("Count")

plt.show()

**Heatmap (Correlation Matrix)**

A heatmap visually represents the correlation between variables.

sns.heatmap(df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)

plt.title("Correlation Heatmap")

plt.show()

**Conclusion**

Pandas provides powerful tools for computing descriptive statistics, summarizing data, and visualizing trends. By integrating with **Matplotlib** and **Seaborn**, you can generate meaningful insights quickly.