pandas

August 31, 2024

[76]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

CREATING PANDAS SERIES AND DATA FRAME

Series

#Creating series using list data=[1,2,3,4,5] pd.Series(data)

[16]:
```
#Using Dictionary
data={1:'a',2:'b',3:'c'}
pd.Series(data)
```

[16]:
```
1    a
2    b
3    c
dtype: object
```

[18]:
```
#Using a Scalar Value
pd.Series(51,index=['a','b','c'])
```

[18]:
```
a    51
b    51
c    51
dtype: int64
```

[26]:
```
#Using a Numpy Array
data=np.random.randint(1,10,10)
pd.Series(data)
```

[26]:
```
0    9
1    3
2    9
3    2
4    7
5    3
6    8
```

```
7    4
8    8
9    1
dtype: int32
```

Data Frame

[41]:
```
#From Dictionary of Lists
data = {
    'Name': ['abc', 'pqr', 'uvw','xyz'],
    'Age': [20,22,24,26],
    'City': ['Vizag', 'Guntur', 'Vijayawada','Tirupathi']
}
pd.DataFrame(data)
```

[41]:

	Name	Age	City
0	abc	20	Vizag
1	pqr	22	Guntur
2	uvw	24	Vijayawada
3	xyz	26	Tirupathi

[43]:
```
#From list of Dictionaries
data = [
    {'Name': 'abc', 'Age': 20, 'City': 'Vizag'},
    {'Name': 'pqr', 'Age': 22, 'City': 'Guntur'},
    {'Name': 'uvw', 'Age': 24, 'City': 'Vijayawada'},
    {'Name': 'xyz', 'Age': 26, 'City': 'Tirupathi'}
]
pd.DataFrame(data)
```

[43]:

	Name	Age	City
0	abc	20	Vizag
1	pqr	22	Guntur
2	uvw	24	Vijayawada
3	xyz	26	Tirupathi

[45]:
```
#from Dictionary of Series
data = {
    'Name': pd.Series(['abc', 'pqr', 'uvw','xyz']),
    'Age': pd.Series([20,22,24,26]),
    'City': pd.Series(['Vizag', 'Guntur', 'Vijayawada','Tirupathi'])
}
pd.DataFrame(data)
```

[45]:

	Name	Age	City
0	abc	20	Vizag
1	pqr	22	Guntur
2	uvw	24	Vijayawada

```
3  xyz   26   Tirupathi
```

[47]:
```
#From list of Lists
data = [
    [20, 'abc', 'Vizag'],
    [22, 'pqr', 'Guntur'],
    [24, 'uvw', 'Vijayawada'],
    [26,'xyz','Tirupathi']
]
pd.DataFrame(data, columns=['Age', 'Name', 'City'])
```

[47]:

	Age	Name	City
0	20	abc	Vizag
1	22	pqr	Guntur
2	24	uvw	Vijayawada
3	26	xyz	Tirupathi

Operations on Pandas

Selcting Columns

[65]:
```
#From Dictionary of Lists
data = {
    'Name': ['abc', 'pqr', 'uvw','xyz'],
    'Age': [20,22,24,26],
    'City': ['Vizag', 'Guntur', 'Vijayawada','Tirupathi']
}
data=pd.DataFrame(data)
```

[54]:
```
data
```

[54]:

	Name	Age	City
0	abc	20	Vizag
1	pqr	22	Guntur
2	uvw	24	Vijayawada
3	xyz	26	Tirupathi

[56]:
```
data["Name"]
```

[56]:
```
0    abc
1    pqr
2    uvw
3    xyz
Name: Name, dtype: object
```

[58]:
```
data[["Name","Age"]]
```

[58]:
```
  Name  Age
0  abc   20
1  pqr   22
2  uvw   24
3  xyz   26
```

Selecting Rows by Index

[96]:
```
data.loc[0]
```

[96]:
```
Name      abc
Age        20
City    Vizag
Name: 0, dtype: object
```

[78]:
```
data.loc[0][1]
```

[78]:
```
20
```

[88]:
```
data.loc[:0]
```

[88]:
```
  Name  Age   City
0  abc   20  Vizag
```

[90]:
```
data.iloc[0]
```

[90]:
```
Name      abc
Age        20
City    Vizag
Name: 0, dtype: object
```

[102]:
```
data.iloc[:2]
```

[102]:
```
  Name  Age    City
0  abc   20   Vizag
1  pqr   22  Guntur
```

Filtering Rows

[105]:
```
data[data["Age"]>22]
```

[105]:
```
  Name  Age        City
2  uvw   24  Vijayawada
3  xyz   26   Tirupathi
```

[109]:
```
data[data["City"]=="Vizag"]
```

[109]:
```
  Name  Age   City
0  abc   20  Vizag
```

[111]:
```
data[data["Age"]>30]
```

[111]:
```
Empty DataFrame
Columns: [Name, Age, City]
Index: []
```

[119]:
```
data[(data["Age"]>28) | (data["City"]=="Tirupathi")]
```

[119]:
```
  Name  Age       City
3  xyz   26  Tirupathi
```

Modify Data

[122]:
```
data["Salary"]=[70000,60000,1000000,800000]
```

[124]:
```
data
```

[124]:
```
  Name  Age        City   Salary
0  abc   20       Vizag    70000
1  pqr   22      Guntur    60000
2  uvw   24  Vijayawada  1000000
3  xyz   26   Tirupathi   800000
```

[130]:
```
data.loc[data["Name"]=="pqr","Name"]="def"
```

[132]:
```
data
```

[132]:
```
  Name  Age        City   Salary
0  abc   20       Vizag    70000
1  def   22      Guntur    60000
2  uvw   24  Vijayawada  1000000
3  xyz   26   Tirupathi   800000
```

Data Handling with Pandas

[135]:
```
df=pd.read_csv("train.csv")
```

[137]:
```
df.sample(3)
```

[137]:
```
     PassengerId  Survived  Pclass  \
200          201         0       3
726          727         1       2
297          298         0       1

                                              Name     Sex   Age  SibSp  Parch  \
200                Vande Walle, Mr. Nestor Cyriel    male  28.0      0      0
726  Renouf, Mrs. Peter Henry (Lillian Jefferys)  female  30.0      3      0
297                 Allison, Miss. Helen Loraine  female   2.0      1      2
```

```
     Ticket    Fare    Cabin Embarked
200  345770    9.50      NaN        S
726   31027   21.00      NaN        S
297  113781  151.55  C22 C26        S
```

[141]:
```
df.isnull().sum()
```

[141]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Filling Missing Data

[174]:
```
#data cleaning
df["Age"].fillna(df["Age"].mean(),inplace=True)
```

[158]:
```
df.isnull().sum()
```

[158]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

[166]:
```
df.info()
```

```
<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          891 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
```

[176]:
```
#Changing Data types
df["Age"]=df["Age"].astype(int)
```

[172]:
```
df.info()
```

```
<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          891 non-null    int32
 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(1), int32(1), int64(5), object(5)
memory usage: 80.2+ KB
```

Data Aanalysis Using Pandas

Grouping and Aggregation

[184]:
```
survive=df.groupby(["Survived"])
```

[186]:
```
survive
```

[186]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001D80C688E00>
```

[206]:
```
survive.get_group(1)["Sex"].value_counts()
```

[206]:
```
Sex
female    233
male      109
Name: count, dtype: int64
```

[230]:
```
df['Ticket'] = pd.to_numeric(df['Ticket'], errors='coerce')
```

[232]:
```
survive["Ticket"].agg(["mean","median"])
```

[232]:

Survived	mean	median
0	305909.648649	315094.0
1	187265.094488	111397.5

[234]:
```
survive["Ticket"].agg(["min","max"])
```

[234]:

Survived	min	max
0	693.0	3101296.0
1	1601.0	3101298.0

Application of Pandas in Data Science

Pandas is a powerful and essential library for data science professionals, offering robust data manipulation and analysis capabilities. In our program, Pandas allows us to efficiently handle, clean, and analyze data, which is crucial for making data-driven decisions. Here's an explanation of how Pandas aids data scientists, its advantages over traditional Python data structures, and real-world applications:

Advantages of Using Pandas Over Traditional Python Data Structures Ease of Data Manipulation:

Pandas provides high-level data manipulation tools such as DataFrame and Series, making it easier to filter, aggregate, and transform data compared to lists or dictionaries in Python. Operations such as merging, joining, pivoting, and reshaping datasets are straightforward in Pandas, whereas they would be cumbersome and error-prone with native Python structures. Handling Missing Data:

Pandas offers built-in methods to handle missing data (fillna, dropna, etc.), making it easy to prepare data for analysis without extensive coding. It also supports time-series data manipulation, allowing for complex operations like resampling and time-shifting, which are not directly available in traditional Python structures. Performance and Efficiency:

Pandas is built on top of NumPy, which provides efficient array-based operations. This results in faster computation and better performance, especially with large datasets. The library is optimized for data manipulation tasks, reducing the need for manual loops and making code more concise and readable. Integration with Other Data Science Tools:

Pandas seamlessly integrates with other data science libraries such as Matplotlib, Seaborn, and Scikit-learn, enabling a smooth workflow from data processing to visualization and modeling. It

also supports various data formats (CSV, Excel, SQL, JSON), making it versatile for data ingestion and export. Real-World Examples of Pandas in Data Science Data Cleaning:

In the real world, data is often messy, containing duplicates, missing values, and inconsistent formats. For example, in financial data analysis, Pandas can be used to clean stock price datasets by handling missing values, ensuring consistent date formats, and filtering out anomalies. Exploratory Data Analysis (EDA):

EDA is a critical step in data science to understand data distribution, detect patterns, and identify relationships between variables. For instance, in healthcare, Pandas can be used to analyze patient data, summarize key statistics, visualize correlations between different health metrics, and prepare the data for predictive modeling. Time Series Analysis:

Pandas is essential for working with time-series data, such as stock market prices, weather data, or sales data. It provides functionality to resample data (e.g., converting daily data to monthly), compute rolling averages, and handle time-zone-aware timestamps, which are pivotal for accurate time-based analysis. Data Aggregation and Grouping:

In scenarios like customer segmentation in marketing, Pandas allows for grouping data by different categories (e.g., demographics, purchase history) to perform aggregated analysis like calculating average spend per segment or identifying high-value customer groups. Data Visualization:

Pandas works hand-in-hand with visualization libraries, allowing quick plotting of data directly from DataFrames. For example, during EDA, a data scientist can quickly generate line plots, histograms, or box plots to visualize trends and distributions, aiding in the hypothesis generation phase. Conclusion Pandas empowers data science professionals to efficiently handle and analyze complex datasets, making it a fundamental tool in the data science toolkit. Its ease of use, integration capabilities, and powerful data manipulation functions make it far superior to traditional Python data structures for data-intensive tasks. Whether cleaning data, conducting EDA, or preparing data for machine learning, Pandas streamlines the entire data analysis pipeline, making it indispensable for any data-driven project.

[ ]: