Pandas Library

Sumedha Kulasekara
Session 3

Pandas library

pandas is a data manipulation package in Python for tabular data. That is, data in the form of rows and columns, also known as DataFrames

pandas is used throughout the data analysis workflow. With pandas, you can:

- Import datasets from databases, spreadsheets, comma-separated values (CSV) files, and more.
- Clean datasets, for example, by dealing with missing values.
- Tidy datasets by reshaping their structure into a suitable format for analysis.
- Aggregate data by calculating summary statistics such as the mean of columns, correlation between them, and more.
- Visualize datasets and uncover insights.

Installation

```
!pip install pandas
```

Importing NumPy

```
import pandas as pd

print(pd.__version__)  # Prints the pandas version
```

Creating a DataFrame from a List of Lists

```
import pandas as pd

# Data as a list of lists
data = [
    ['John', 28, 'New York', 50000],
    ['Sara', 22, 'Los Angeles', 60000],
    ['Paul', 35, 'Chicago', 55000],
    ['Anna', 30, 'Houston', 65000]
]

# Column names
columns = ['Name', 'Age', 'City', 'Salary']

# Creating DataFrame
df = pd.DataFrame(data, columns=columns)

print(df)
```

Creating a DataFrame from a CSV File

```
import pandas as pd

# Read data from a CSV file
customer_data = pd.read_csv('customer_data.csv')
print(customer_data )
```

Row(s) to use as column names. By default, Pandas assumes that the first row contains the column names. You can specify a different row or set it to None if the CSV does not contain headers.

```
df = pd.read_csv('data.csv', header=0)  # First row is the header (default)
df = pd.read_csv('data.csv', header=None)  # No header row
```

“names” allows you to specify a list of column names if the CSV doesn't contain headers or if you want to override them.input it as a list.

```
df = pd.read_csv('data.csv', names=['Column1', 'Column2', 'Column3'])
```

Creating a DataFrame from a txtFile

```
# Read the space-separated text file into a DataFrame
diabetes = pd.read_csv("diabetes.txt", sep="\s")

# Display the DataFrame
print(diabetes)
```

Importing Excel files (single sheet)

```
!pip install openpyxl
df = pd.read_excel('diabetes.xlsx')
```

Importing Excel files (multiple sheets)

```
# Extracting the second sheet since Python uses 0-indexing
df = pd.read_excel('diabetes_multi.xlsx', sheet_name=1)
```

view data using .head() and .tail()

You can view the first few or last few rows of a DataFrame using the .head() or .tail() methods, respectively. You can specify the number of rows through the n argument (the default value is 5).

```
df= pd.read_csv('customer_data.csv')
# df= dataset name
df.head()
df.head(10)

df.tail()
df.tail(n = 10)
```

Understanding data using .describe()

The .describe() method prints the summary statistics of all numeric columns, such as count, mean, standard deviation, range, and quartiles of numeric columns.

```
# df= dataset name
df.describe()
df.describe().T
```

Understanding data using .info()

The .info() method is a quick way to look at the data types, missing values, and data size of a DataFrame.

```
# df= dataset name
df.info()
```

Understanding your data using .shape

The number of rows and columns of a DataFrame can be identified using the .shape attribute of the DataFrame. It returns a tuple (row, column) and can be indexed to get only rows, and only columns count as output.

```
# df= dataset name
df.shape # Get the number of rows and columns
df.shape[0] # Get the number of rows only
df.shape[1] # Get the number of columns only
```

Get all columns and column names

```
# df= dataset name
df.columns
```

Copy of the original DataFrame

.copy() method makes a copy of the original DataFrame.

```
# df= dataset name
df2 = df.copy()
```

Inspect Data

- df.head() #View the first 5 rows of the DataFrame.
- df.tail() #View the last 5 rows of the DataFrame.
- df.sample() #View the random 5 rows of the DataFrame.
- df.shape #Get the dimensions of the DataFrame.
- df.info() #Get a concise summary of the DataFrame.
- df.describe()#Summary statistics for numerical columns.
- df.dtypes #Check data types of columns.
- df.columns #List column names.
- df.index #Display the index range.

Select Index Data

- df['Age'] #Select a single column
- df[['Age', 'Income']] #Selectmultiple columns.
- df.iloc[0] #Select the first row by position.
- df.loc[0] #Select the first row by index label.
- df.iloc[0, 0] #Select a specific element by position.
- df.loc[0, 'Age'] #Select a specific element by label.
- df[df['Age'] > 50] #Filter rows where column > 5.
- df.iloc[0:5, 0:2] #Slice rows and columns.
- df.set_index('CustomerID') #Set a column as the index.

Cleaning Data

- df.isnull() #Check for null values.
- df.isnull().sum()

- df.notnull() #Check for non-null values..

- df['Income'] = df['Income'].fillna(df['Income'].mean()) # Replace NaN in 'Income' with mean value
- df['Income'].fillna(0).head() #Replace null values with a specific value.

- df.dropna() #Drop rows with null values.

- df['Subscription'].replace('Basic', 'Standard').head() #Replace specific values.

- df.rename(columns={'Age': 'CustomerAge'}).head()) # Rename columns.

- df['Age'].astype('float').head() #Change data type of a column..

- df.drop_duplicates() #Remove duplicate rows

- df.reset_index() # Reset the index.

Sort Filter Data

- df.sort_values('Age') #Sort by column in ascending order.
- df.sort_values('Age', ascending=False) #Sort by column in descending order.
- df.sort_values(['City','Age'], ascending=[True, False]) #Sort by multiple columns.
- df[df['Age'] > 50] #Filter rows based on condition.
- df.query('Age > 50') #Filter using a query string.
- df.sample(5, random_state=1) #Randomly select 5 rows.
- df.nlargest(3, 'Income') #Get top 3 rows by column.
- df.nsmallest(3, 'Income') #Get bottom 3 rows by column.
- df.filter(like='Score') #Filter columns by substring.(returns only the LoyaltyScore column.)
- print("Unique Count:", df['Gender'].nunique()) # Unique count for Gender
- print(df['Gender'].value_counts())# Value counts for Gender

Group Data

- df.groupby('Subscription') # Group by a column
- df.groupby('Subscription').mean(numeric_only=True) # Mean of groups
- df.groupby('Subscription').sum(numeric_only=True) # Sum of groups
- df.groupby('Subscription').count() # Count non-null values in groups
- df.groupby('Subscription')['Age'].max() # Max Age for each Subscription
- df.pivot_table(values='PurchaseAmount', index='Subscription', columns='City', aggfunc='mean') #Pivot table (mean PurchaseAmount by Subscription & City)
- df.agg({'Age': 'mean', 'PurchaseAmount': 'sum'}) # Aggregate multiple columns
- df.select_dtypes(include=[np.number]).apply(np.mean) # Apply mean to numeric columns
- df['Age'].transform(lambda x: x + 10) # Transform (Age + 10)

Statistical Operations

- numeric_df = df.select_dtypes(include=['number']) # Filter for numeric columns only
- numeric_df.mean() #Column-wise mean.
- numeric_df.median() # Column-wise median.
- numeric_df.std() #Column-wise standard deviation.
- numeric_df.var() #Column-wise variance.
- numeric_df.sum() #Column-wise sum.
- numeric_df.min() #Column-wise minimum.
- numeric_df.max() #Column-wise maximum.
- numeric_df.count() #Count of non-null values per column.
- numeric_df.corr() #Correlation matrix.

Data Visualization

```
# Line plot
df['Age'].plot(kind='line')
plt.title("Line Plot of Age")
plt.xlabel("Index")
plt.ylabel("Age")
plt.show()

# Bar plot
df['Subscription'].value_counts().plot(kind='bar')

# Horizontal bar plot
df['Subscription'].value_counts().plot(kind='barh')

# Histogram
df['Income'].plot(kind='hist', bins=20)

# Box plot
df['Income'].plot(kind='box')
```

Data Visualization

```
# KDE plot
df['Income'].plot(kind='kde')

# Pie chart
df['Subscription'].value_counts().plot(kind='pie', autopct='%1.1f%%')

# Scatter plot
df.plot.scatter(x='Age', y='Income')

# Area plot
df[['Age','Income_filled','PurchaseAmount']].head(50).plot(kind='area', alpha=0.5)
```

Data Visualization

Summary of Plots and Data Types:

- **Numerical Data: Line, histogram, box, KDE, scatter, area plots.**
- **Categorical Data: Bar, horizontal bar, pie plots.**
- **Bivariate (Two Variables): Scatter plot (for relationships between two numerical variables).**
- **Trend or Time Series: Line plot (especially useful for time series).**

Missing values in pandas with .isnull()

You can combine .isnull() with .sum() to count the number of nulls in each column.

```
df.isnull().sum()
```

Sorting, Slicing and Extracting Data in pandas

To sort a DataFrame by a specific column:

```
df.sort_values(by="Age", ascending=False, inplace=True)  # Sort by Age in descending order
df.sort_values(by=["Age", "Income"], ascending=[False, True], inplace=True)
```

(inplace=True means modify the original DataFrame)

If you filter or sort a DataFrame, your index might become misaligned. Use `.reset_index()` to fix this:If you filter or sort a DataFrame, your index might become misaligned. Use .reset_index() to fix this:

```
df.reset_index(drop=True, inplace=True)  # Resets index and removes old index column
df.head()
```

Isolating Columns

df['Age'] df[[‘Age’,‘Income’]]

Isolating Raws

df[df.index==1] #the second row with index = 1 is returned

df[df.index.isin(range(2,10))] #2 - 9 raws

loc:It is label-based, meaning it uses the row and column names (labels) to make selections. With loc, the slicing is inclusive, including both the start and end labels.

df.loc[1]#Row with index labeled 1

df.loc[100:110]

Conditional Slicing

```
df[df.Age == 32]
df[df.Subscription == 'VIP']
```

Adding New Column to Existing DataFrame in Pandas

```
# Define a dictionary containing Students data
data = {'Name': ['John', 'David', 'Rafael', 'Rodrick'],
        'Height': [145, 162, 158, 163],
        'Qualification': ['A', 'B', 'C', 'D']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# Declare a list that is to be converted into a column
address = ['NewYork', 'Texas', 'Colorado', 'Dakota']

# Using 'Address' as the column name and equating it to the list
df['Address'] = address

display(df)
```

Delete rows/columns from DataFrame using Pandas.drop()

```
# Drop rows based on Name column values
table = table[~table['Name'].isin(['David', 'Rafael'])]

# Display the DataFrame after dropping rows
print(table)

# dropping passed values
table.drop(["Address"],axis = 1, inplace = True)

# display
print(table)
```

Exercise

1. Read in the csv file (csv) Lands.txt. Create a table with the columns 'land', 'area', 'female', 'male', 'population' and 'density' (inhabitants per square kilometres).

   population = male + female

   density = population* 1000 / area

2. print out the rows where the area is greater than “30000” and the population is greater than “10500”.
3. Print the rows where the density is greater than 250.

Arithmetic Operations in pandas dataframe

```
import pandas as pd

# Create a sample DataFrame
data = {'A': [10, 20, 30, 40], 'B': [0.5, 0.6, 0.7, 0.8]}
df = pd.DataFrame(data)

# Display the input DataFrame
print("Input DataFrame:\n", df)

# Perform arithmetic operations
print("\nAddition:\n", df + 2)
print("\nSubtraction:\n", df - 2)
print("\nMultiplication:\n", df * 2)
print("\nDivision:\n", df / 2)
print("\nExponentiation:\n", df ** 2)
print("\nModulus:\n", df % 2)
print("\nFloor Division:\n", df // 2)
```

Data Normalization with Pandas

Data Normalization could also be a typical practice in machine learning which consists of transforming numeric columns to a standard scale.

```
# importing packages
import pandas as pd

# create data
df = pd.DataFrame([
                [180000, 110, 18.9, 1400],
                [360000, 905, 23.4, 1800],
                [230000, 230, 14.0, 1300],
                [60000, 450, 13.5, 1500]],

                columns=['Col A', 'Col B',
                        'Col C', 'Col D'])
# view data
display(df)

import matplotlib.pyplot as plt
df.plot(kind = 'bar')
```

Using The maximum absolute scaling

The maximum absolute scaling rescales each feature between -1 and 1 by dividing every observation by its maximum absolute value. We can apply the maximum absolute scaling in Pandas using the .max() and .abs() methods, as shown below.

```
# copy the data
df_max_scaled = df.copy()

# apply normalization techniques
for column in df_max_scaled.columns:
    df_max_scaled[column] = df_max_scaled[column]  / df_max_scaled[column].abs().max()

# view normalized data
display(df_max_scaled)


import matplotlib.pyplot as plt
df_max_scaled.plot(kind = 'bar')
```

Python Pandas - Working with Text Data

Creating DataFrames with Text Data
Changing Case
String Length:
Replacing Text:
Splitting Text:
Extracting Substrings:
Regular Expressions (Regex) with Text Data
Handling Missing Data in Text Columns
Text Data Aggregation and Grouping