Pandas

3.data\_structures

Poornima (Guest)9:46 AM

'''  
Panel Data  
Panel data, also known as longitudinal data or cross-sectional time series data,   
involves observations of multiple phenomena obtained over multiple time periods   
for the same firms or individuals. In pandas, Panel data used to be handled using   
the Panel class, but it has since been deprecated in favor of using multi-index DataFrames.  
  
Panel Data Structure  
Panel data structure allows for the storage and manipulation of three-dimensional data,  
 typically with dimensions (items, major\_axis, minor\_axis):  
  
Items: Axis 0, each item corresponds to a DataFrame (like different variables).  
Major\_axis: Axis 1, usually represents time.  
Minor\_axis: Axis 2, represents individual entities (like different firms or individuals).  
Due to the deprecation of Panel, we now use multi-index DataFrames to handle panel data.  
  
Creating and Manipulating Panels  
Creating Panel-like Data with Multi-index DataFrames  
'''  
  
import pandas as pd  
import numpy as np  
  
# Create a multi-index DataFrame  
arrays = [  
 ['A', 'A', 'B', 'B'],  
 [1, 2, 1, 2]  
]  
  
index = pd.MultiIndex.from\_arrays(arrays, names=('person', 'time'))  
data = pd.DataFrame(np.random.randn(4, 3), index=index, columns=['entity1',  
 'entity2', 'entity3'])  
  
print("Multi-index DataFrame:")  
print(data)  
  
#Manipulating Multi-index DataFrames  
# Access data for variable 'A'  
print("\nData for variable 'A':")  
print(data.loc['A'])  
  
# Access data for time period 1  
print("\nData for time period 1:")  
print(data.xs(1, level='time'))  
  
# Adding a new row for a new time period  
new\_data = pd.DataFrame({  
 'entity1': [0.5, 0.3],  
 'entity2': [1.5, 1.3],  
 'entity3': [2.5, 2.3]  
}, index=pd.MultiIndex.from\_product([['A', 'B'], [3]], names=['variable', 'time']))  
print(new\_data)  
  
data = pd.concat([data, new\_data])  
print("\nData after adding new time period:")  
print(data)  
  
'''  
Applications and Use Cases  
  
Economics and Finance: Analyzing the financial performance of different firms over time.  
Healthcare: Monitoring patient health metrics across different time periods.  
Social Sciences: Studying the behavior of individuals across various time points.  
Marketing: Observing the impact of marketing campaigns over time.  
  
'''

4.data\_manipulation

[10:14 AM] Poornima (Guest)

'''  
Data Manipulation with Pandas  
'''  
  
#Reading Data from CSV, Excel, JSON, and SQL  
  
import pandas as pd  
  
# Creating a sample DataFrame  
data = pd.DataFrame({  
 'A': [1, 2, 3, 4],  
 'B': [5, 6, 7, 8],  
 'C': [9, 10, 11, 12]  
})  
  
data.to\_csv('sample\_data.csv', index=False)  
data.to\_excel('sample\_data.xlsx', index=False)  
data.to\_json('sample\_data.json', orient='records')  
  
  
# Reading data  
  
csv\_data = pd.read\_csv('sample\_data.csv')  
print("Data from CSV:\n", csv\_data)  
  
excel\_data = pd.read\_excel('sample\_data.xlsx')  
print("Data from Excel:\n", excel\_data)  
  
json\_data = pd.read\_json('sample\_data.json')  
print("Data from JSON:\n", json\_data)  
  
  
import sqlite3  
# Create an in-memory SQLite database and insert the sample data  
conn = sqlite3.connect(':memory:')  
data.to\_sql('sample\_table', conn, index=False, if\_exists='replace')  
  
# Reading data from SQL  
sql\_data = pd.read\_sql('SELECT \* FROM sample\_table', conn)  
print("Data from SQL:\n", sql\_data)  
  
  
'''  
  
#==========================================================================

[10:37 AM] Poornima (Guest)

#Selecting Data by Labels (.loc)  
  
import pandas as pd  
  
# Sample DataFrame  
data = pd.DataFrame({  
 'A': [1, 2, 3, 4],  
 'B': [5, 6, 7, 8],  
 'C': [9, 10, 11, 12]  
}, index=['row1', 'row2', 'row3', 'row4'])  
print(data)  
# Selecting rows with label 'row2' and specific columns 'A' and 'C'  
selected\_data = data.loc['row2', ['A', 'C']]  
print(selected\_data)  
  
# Selecting the first 2 rows and first 2 columns  
selected\_data = data.iloc[0:2, 0:2]  
print(selected\_data)  
  
# Selecting rows where column 'A' is greater than 2  
filtered\_data = data[data['A'] > 2]  
print(filtered\_data)

[11:58 AM] Poornima (Guest)

'''  
Data Cleaning  
Handling Missing Data (dropna, fillna)  
'''  
  
# Sample DataFrame with missing values  
data = pd.DataFrame({  
 'A': [1, 2, None, 4],  
 'B': [None, 2, 3, 4],  
 'C': [1, 2, 3, None]  
})  
  
# Dropping rows with any missing values  
cleaned\_data\_drop = data.dropna()  
print('cleaned \n',cleaned\_data\_drop)  
  
# Filling missing values with 0  
cleaned\_data\_fill = data.fillna(0)  
print(cleaned\_data\_fill)  
  
# Sample DataFrame with duplicates  
data = pd.DataFrame({  
 'A': [1, 2, 2, 4],  
 'B': [5, 6, 6, 8]  
})  
  
# Removing duplicate rows  
cleaned\_data = data.drop\_duplicates()  
print(cleaned\_data)  
  
# Sample DataFrame  
data = pd.DataFrame({  
 'A': ['1', '2', '3', '4']  
})  
print('before \n',data)  
print(data.dtypes)  
# Converting data type of column 'A' to integer  
data['A'] = data['A'].astype(int)  
print('after \n',data)  
print(data.dtypes)  
  
# Sample DataFrame  
data = pd.DataFrame({  
 'A': ['Hello', 'World', 'Pandas', 'Python']  
})  
  
# Converting column 'A' to lowercase  
data['A'] = data['A'].str.lower()  
print(data)  
  
  
'''  
Data Transformation  
Applying Functions to Data (apply, map, applymap)  
'''  
  
# Sample DataFrame  
data = pd.DataFrame({  
 'A': [1, 2, 3, 4],  
 'B': [5, 6, 7, 8]  
})  
  
# Applying a function to column 'A'  
data['A'] = data['A'].apply(lambda x: x \* 2)  
print(data)  
  
# Mapping values in column 'B'  
data['B'] = data['B'].map({5: 'Five', 6: 'Six', 7: 'Seven', 8: 'Eight'})  
print(data)  
  
# Applying a function to the entire DataFrame using map  
data = data.applymap(str)  
print('str tra',data)  
print(data.dtypes)  
  
# Sample DataFrame  
data = pd.DataFrame({  
 'A': [3, 1, 4, 2],  
 'B': [5, 6, 7, 8]  
})  
  
# Sorting data by column 'A'  
sorted\_data = data.sort\_values(by='A')  
print(sorted\_data)  
  
data = pd.DataFrame({  
 'A': [3, 1, 2, 2],  
 'B': [5, 6, 7, 8]  
})  
  
# Ranking data in column 'A'  
data['rank'] = data['A'].rank()  
print(data)  
  
# Sample DataFrame  
data = pd.DataFrame({  
 'A': [5, 15, 25, 35, 45, 4, 44,23,38,24]  
})  
  
# Binning data into discrete intervals  
bins = [0, 10, 20, 30, 40, 50]  
labels = ['0-10', '10-20', '20-30', '30-40', '40-50']  
data['binned'] = pd.cut(data['A'], bins=bins, labels=labels)  
print(data)

5.adv\_data\_operations

[12:45 PM] Poornima (Guest)

'''  
Merging and Joining DataFrames  
'''  
  
import pandas as pd  
  
# Creating sample DataFrames  
df1 = pd.DataFrame({  
 'A': ['A0', 'A1', 'A2', 'A3'],  
 'B': ['B0', 'B1', 'B2', 'B3'],  
 'C': ['C0', 'C1', 'C2', 'C3'],  
 'D': ['D0', 'D1', 'D2', 'D3']  
}, index=[0, 1, 2, 3])  
  
df2 = pd.DataFrame({  
 'A': ['A4', 'A5', 'A6', 'A7'],  
 'B': ['B4', 'B5', 'B6', 'B7'],  
 'C': ['C4', 'C5', 'C6', 'C7'],  
 'D': ['D4', 'D5', 'D6', 'D7']  
}, index=[4, 5, 6, 7])  
  
# Concatenating DataFrames  
result = pd.concat([df1, df2])  
print("Concatenated DataFrame:\n", result)  
  
#Merging on keys (merge)  
  
left = pd.DataFrame({  
 'key': ['K0', 'K1', 'K2', 'K3'],  
 'A': ['A0', 'A1', 'A2', 'A3'],  
 'B': ['B0', 'B1', 'B2', 'B3']  
})  
  
right = pd.DataFrame({  
 'key': ['K0', 'K1', 'K4', 'K5'],  
 'C': ['C0', 'C1', 'C2', 'C3'],  
 'D': ['D0', 'D1', 'D2', 'D3']  
})  
  
# Merging DataFrames  
result = pd.merge(left, right, on='key')  
print("Merged DataFrame:\n", result)  
  
#Joining DataFrames (join)  
# Creating sample DataFrames with different indexes  
dfj1 = pd.DataFrame({  
 'A': ['A0', 'A1', 'A2'],  
 'B': ['B0', 'B1', 'B2']  
}, index=['K0', 'K1', 'K2'])  
  
dfj2 = pd.DataFrame({  
 'C': ['C0', 'C2', 'C3'],  
 'D': ['D0', 'D2', 'D3']  
}, index=['K0', 'K2', 'K3'])  
  
# Joining DataFrames  
result = dfj1.join(dfj2, how='outer')  
print("Joined DataFrame:\n", result)  
  
  
#Grouping and Aggregation  
#Grouping data (groupby)  
  
import pandas as pd  
import numpy as np  
  
# Creating a sample DataFrame  
df = pd.DataFrame({  
 'A': ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],  
 'B': ['one', 'one', 'two', 'two', 'one', 'one', 'two', 'two'],  
 'C': [1, 2, 3, 4, 5, 6, 7, 8],  
 'D': np.random.randn(8)  
})  
  
# Grouping by column 'A'  
grouped = df.groupby('A')  
# Aggregating numeric data with mean  
mean\_result = grouped[['C', 'D']].mean()  
print("Grouped DataFrame mean:\n", mean\_result)

[2:45 PM] Poornima (Guest)

#Time Series Analysis  
#Working with date and time data  
  
# Creating a sample DataFrame with date range  
dates = pd.date\_range('20230101', periods=6)  
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))  
print("DataFrame with dates:\n", df)  
  
#Resampling and frequency conversion  
# Resampling to a different frequency  
resampled = df.resample('M').mean()  
print("Resampled DataFrame (monthly mean):\n", resampled)  
  
#Rolling and expanding windows  
# Applying rolling window  
rolling = df.rolling(window=2).mean()  
print("Rolling window (mean):\n", rolling)  
  
# Applying expanding window  
expanding = df.expanding(min\_periods=1).mean()  
print("Expanding window (mean):\n", expanding)

Poornima (Guest)3:03 PM

#Visualization with Pandas  
#Basic plotting with Pandas  
  
import matplotlib.pyplot as plt  
  
# Creating a sample DataFrame  
df = pd.DataFrame(np.random.randn(10, 4), columns=['A', 'B', 'C', 'D'])  
  
# Plotting  
df.plot()  
plt.title('Basic Plot')  
plt.show()  
  
# Customizing plot with labels and title  
df.plot()  
plt.xlabel('X-axis label')  
plt.ylabel('Y-axis label')  
plt.title('Customized Plot')  
plt.show()  
  
#Integration with Matplotlib and Seaborn  
import seaborn as sns  
  
# Using Seaborn for a more advanced plot  
sns.lineplot(data=df)  
plt.title('Seaborn Line Plot')  
plt.show()

Data processing

1.data\_proc.py

[3:39 PM] Poornima (Guest)

#Missing values  
import pandas as pd  
import numpy as np  
  
# Creating a sample DataFrame with missing values  
data = {  
 'A': [1, 2, np.nan, 4, 5],  
 'B': [np.nan, 2, 3, np.nan, 5],  
 'C': [1, 2, 3, 4, 5],  
 'D': [np.nan, np.nan, np.nan, 4, 5]  
}  
df = pd.DataFrame(data)  
print('Data \n', df)  
# Identifying missing data  
print("Count of Missing data:\n", df.isnull().sum())  
  
#Techniques to Handle Missing Data  
  
'''  
data = {  
 'A': [1, 2, np.nan, 4, 5],  
 'B': [np.nan, 2, 3, np.nan, 5],  
}  
df = pd.DataFrame(data)  
'''  
# Dropping rows with any missing data  
df\_dropped\_rows = df.dropna(axis=0)  
print("DataFrame after dropping rows with any missing data:\n", df\_dropped\_rows)  
  
# Dropping columns with any missing data  
df\_dropped\_cols = df.dropna(axis=1)  
print("DataFrame after dropping columns with any missing data:\n", df\_dropped\_cols)  
  
# Filling missing data with a specific value  
df\_filled = df.fillna(0)  
print("DataFrame after filling missing data with 0:\n", df\_filled)  
  
'''  
Imputation Methods  
'''  
  
# Imputing missing data with the mean of each column  
df\_mean\_imputed = df.fillna(df.mean())  
print("DataFrame after mean imputation:\n", df\_mean\_imputed)  
  
# Imputing missing data with the median of each column  
df\_median\_imputed = df.fillna(df.median())  
print("DataFrame after median imputation:\n", df\_median\_imputed)  
  
# Imputing missing data with the mode of each column  
df\_mode\_imputed = df.fillna(df.mode().iloc[0])  
print("DataFrame after mode imputation:\n", df\_mode\_imputed)  
  
# Interpolating missing data  
df\_interpolated = df.interpolate()  
print("DataFrame after interpolation:\n", df\_interpolated)

2.normalization\_scaling

[4:28 PM] Poornima (Guest)

import pandas as pd  
from sklearn.preprocessing import StandardScaler  
  
#StandardScaler  
# Creating a sample DataFrame  
data = {  
 'Feature1': [1, 2, 3, 4, 5],  
 'Feature2': [10, 20, 30, 40, 50],  
 'Feature3': [100, 200, 300, 400, 500]  
}  
df = pd.DataFrame(data)  
  
# Applying standardization  
scaler = StandardScaler()  
df\_standardized = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)  
  
print("Standardized DataFrame:\n", df\_standardized)

[4:45 PM] Poornima (Guest)

#MinMaxScaler  
from sklearn.preprocessing import MinMaxScaler  
  
# Applying min-max scaling  
scaler = MinMaxScaler()  
df\_minmax\_scaled = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)  
  
print("Min-Max Scaled DataFrame:\n", df\_minmax\_scaled)  
  
  
#RobustScaler  
from sklearn.preprocessing import RobustScaler  
  
# Applying robust scaling  
scaler = RobustScaler()  
df\_robust\_scaled = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)  
  
print("Robust Scaled DataFrame:\n", df\_robust\_scaled)

3.encoding\_categorical\_values:

[5:30 PM] Poornima (Guest)

import pandas as pd  
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, OrdinalEncoder  
  
# One-Hot Encoding Example  
data\_one\_hot = {  
 'City': ['New York', 'Paris', 'Berlin', 'New York', 'Berlin']  
}  
df\_one\_hot = pd.DataFrame(data\_one\_hot)  
print(df\_one\_hot)  
  
# Using pandas get\_dummies for One-Hot Encoding  
df\_one\_hot\_pd = pd.get\_dummies(df\_one\_hot, columns=['City'])  
print("One-Hot Encoded DataFrame using pandas:\n", df\_one\_hot\_pd)  
  
# Using sklearn's OneHotEncoder  
encoder = OneHotEncoder(sparse\_output=False)  
one\_hot\_encoded = encoder.fit\_transform(df\_one\_hot[['City']])  
print('before \n',one\_hot\_encoded)  
df\_one\_hot\_sklearn = pd.DataFrame(one\_hot\_encoded,  
 columns=encoder.get\_feature\_names\_out(['City']))  
print("One-Hot Encoded DataFrame using sklearn:\n", df\_one\_hot\_sklearn)

[9:35 AM] Poornima (Guest)

# Label Encoding Example  
data\_label = {  
 'City': ['New York', 'Paris', 'Berlin', 'New York', 'Berlin']  
}  
df\_label = pd.DataFrame(data\_label)  
  
# Applying LabelEncoder  
label\_encoder = LabelEncoder()  
df\_label['City\_Label'] = label\_encoder.fit\_transform(df\_label['City'])  
print("Label Encoded DataFrame:\n", df\_label)  
  
# Ordinal Encoding Example  
data\_ordinal = {  
 'Size': ['Small', 'Medium', 'Large', 'Medium', 'Small']  
}  
df\_ordinal = pd.DataFrame(data\_ordinal)  
  
# Applying OrdinalEncoder  
ordinal\_encoder = OrdinalEncoder(categories=[['Small', 'Medium', 'Large']])  
df\_ordinal['Size\_Ordinal'] = ordinal\_encoder.fit\_transform(df\_ordinal[['Size']])  
print("Ordinal Encoded DataFrame:\n", df\_ordinal)

4.feature\_engineering.py

[10:27 AM] Poornima (Guest)

import pandas as pd  
  
#Creating New Features  
# Sample data  
data = {  
 'Order Date': ['2023-01-01', '2023-01-02', '2023-01-03'],  
 'Delivery Date': ['2023-01-05', '2023-01-06', '2023-01-07'],  
 'Product Price': [100, 150, 200]  
}  
  
df = pd.DataFrame(data)  
print('ODF \n', df)  
# Converting columns to datetime  
df['Order Date'] = pd.to\_datetime(df['Order Date'])  
df['Delivery Date'] = pd.to\_datetime(df['Delivery Date'])  
  
# Creating new feature 'Delivery Time'  
df['Delivery Time'] = (df['Delivery Date'] - df['Order Date']).dt.days  
  
print('AFE \n',df)  
  
#Polynomial Features -- Housing Price Prediction  
  
import pandas as pd  
from sklearn.preprocessing import PolynomialFeatures  
  
# Sample data  
data = {  
 'Size (sq ft)': [1500, 2050, 2400],  
 'Number of Bedrooms': [3, 4, 5]  
}  
  
df = pd.DataFrame(data)  
  
# Creating polynomial features  
poly = PolynomialFeatures(degree=2, include\_bias=False)  
poly\_features = poly.fit\_transform(df)  
  
# Creating a DataFrame with the polynomial features  
poly\_df = pd.DataFrame(poly\_features, columns=poly.get\_feature\_names\_out(df.columns))  
  
print(poly\_df)  
  
#Interaction Features -- Customer Demographics and Purchase Behavior  
  
import pandas as pd  
  
# Sample data  
data = {  
 'Age': [25, 35, 45],  
 'Income': [50000, 75000, 100000]  
}  
  
df = pd.DataFrame(data)  
  
# Creating interaction feature 'Age \* Income'  
df['Age \* Income'] = df['Age'] \* df['Income']  
  
print(df)

Visualization

1.simple\_plots.py

[11:40 AM] Poornima (Guest)

import matplotlib.pyplot as plt  
  
# Sample data  
x = [1, 2, 3, 4, 5]  
y = [2, 3, 5, 7, 11]  
  
# Create a line plot  
plt.plot(x, y, marker='o')  
plt.title("Line Plot")  
plt.xlabel("X-axis")  
plt.ylabel("Y-axis")  
  
plt.show()  
  
  
  
  
# Sample data  
categories = ['A', 'B', 'C', 'D']  
values = [10, 15, 7, 10]  
  
# Create a bar plot  
plt.bar(categories, values, color='skyblue')  
plt.title("Bar Plot")  
plt.xlabel("Categories")  
plt.ylabel("Values")  
plt.show()  
  
  
  
# Sample data  
x = [1, 2, 3, 4, 5]  
y = [2, 3, 5, 7, 11]  
  
# Create a scatter plot  
plt.scatter(x, y, color='red')  
plt.title("Scatter Plot")  
plt.xlabel("X-axis")  
plt.ylabel("Y-axis")  
plt.show()

Poornima (Guest)11:51 AM

# Sample data  
x = [1, 2, 3, 4, 5]  
y1 = [2, 3, 5, 7, 11]  
y2 = [1, 4, 6, 8, 10]  
  
# Create a plot with customizations  
plt.plot(x, y1, marker='o', label='Series 1', color='blue')  
plt.plot(x, y2, marker='x', label='Series 2', color='green')  
  
# Customizing the plot  
plt.title("Customized Plot")  
plt.xlabel("X-axis")  
plt.ylabel("Y-axis")  
plt.legend()  
plt.grid(True)  
plt.show()

2.adv\_plots.py

[12:45 PM] Poornima (Guest)

import matplotlib.pyplot as plt  
import numpy as np  
  
  
# Sample data  
x = np.linspace(0, 10, 100)  
y1 = np.sin(x)  
y2 = np.cos(x)  
  
# Create a figure with two subplots  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))  
  
# First subplot  
ax1.plot(x, y1, color='blue', label='Sine')  
ax1.set\_title('Sine Function')  
ax1.set\_xlabel('X-axis')  
ax1.set\_ylabel('Y-axis')  
ax1.legend()  
  
# Second subplot  
ax2.plot(x, y2, color='red', label='Cosine')  
ax2.set\_title('Cosine Function')  
ax2.set\_xlabel('X-axis')  
ax2.set\_ylabel('Y-axis')  
ax2.legend()  
  
# Show the plots  
plt.tight\_layout()  
plt.show()  
  
  
  
# Sample data  
data = np.random.randn(100)  
  
# Create a figure with a histogram and a density plot  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))  
  
# Histogram  
ax1.hist(data, bins=10, color='skyblue', edgecolor='black')  
ax1.set\_title('Histogram')  
ax1.set\_xlabel('Value')  
ax1.set\_ylabel('Frequency')  
  
# Density plot  
ax2.hist(data, bins=10, density=True, color='skyblue', edgecolor='black', alpha=0.6)  
data\_density = np.linspace(min(data), max(data), 100)  
ax2.plot(data\_density, (1/(np.sqrt(2 \* np.pi))) \* np.exp(-0.5 \* (data\_density)\*\*2), color='red')  
ax2.set\_title('Density Plot')  
ax2.set\_xlabel('Value')  
ax2.set\_ylabel('Density')  
  
# Show the plots  
plt.tight\_layout()  
plt.show()  
  
  
  
# Sample data  
data1 = np.random.normal(0, 1, 100)  
data2 = np.random.normal(1, 2, 100)  
data = [data1, data2]  
  
# Create a figure with a box plot and a violin plot  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))  
  
# Box plot  
ax1.boxplot(data, patch\_artist=True)  
ax1.set\_title('Box Plot')  
ax1.set\_xlabel('Data Sets')  
ax1.set\_ylabel('Value')  
  
# Violin plot  
ax2.violinplot(data, showmeans=False, showmedians=True)  
ax2.set\_title('Violin Plot')  
ax2.set\_xlabel('Data Sets')  
ax2.set\_ylabel('Value')  
  
# Show the plots  
plt.tight\_layout()  
plt.show()