WEEK-5

WORKING WITH DATA PART-I

1. Reading and writing a Text Files

with open("example.txt", "w") as file:

# Write content to the file

file.write("Hello, this is a sample text.\n")

file.write("Writing to a file in Python is easy!\n")

file.write("You can write multiple lines as well.\n")

print("File 'example.txt' has been written.")

# Open a file in read mode ("r")

with open("example.txt", "r") as file:

# Read the entire content of the file

content = file.read()

# Alternatively, you can read line by line

# lines = file.readlines()

# Print the content of the file

print("Content of the file:")

print(content)

1. JSON with Python

import json

# Sample data

data = {

"name": "John",

"age": 30,

"city": "New York"

}

# Open a file in write mode ("w") for writing JSON

with open("example.json", "w") as json\_file:

# Write the JSON data to the file

json.dump(data, json\_file)

print("JSON data has been written to 'example.json'.")

import json

# Open a file in read mode ("r") for reading JSON

with open("example.json", "r") as json\_file:

# Load the JSON data from the file

loaded\_data = json.load(json\_file)

# Print the loaded JSON data

print("Loaded JSON data:")

print(loaded\_data)

HTML with PythonMicrosoft

pip install beautifulsoup4 requests

import requests from bs4

import BeautifulSoup

# URL of the website you want to fetch HTML from

url = "https://www.example.com"

# Send a GET request to the URL

response = requests.get(url)

# Check if the request was successful (status code 200)

if response.status\_code == 200:

# Parse the HTML content using BeautifulSoup

soup = BeautifulSoup(response.text, 'html.parser')

# Print the HTML content or perform further processing

print(soup.prettify())

else:

print(f"Failed to fetch HTML. Status code: {response.status\_code}")

# HTML content to write to the file

html\_content = """

<!DOCTYPE html>

<html>

<head>

<title>My HTML Page</title>

</head>

<body>

<h1>Hello, HTML!</h1>

<p>This is a simple HTML page written with Python.</p>

</body>

</html>

"""

# Write HTML content to a file

with open("output.html", "w") as html\_file:

html\_file.write(html\_content)

print("HTML content has been written to 'output.html'.")

Excel file with Python

import pandas as pd

# Read an Excel file

df = pd.read\_excel("example.xlsx")

# Display the DataFrame (contents of the Excel file)

print("DataFrame from Excel:")

print(df)

# Create a sample DataFrame

data = {

'Name': ['John', 'Alice', 'Bob'],

'Age': [25, 30, 22],

'City': ['New York', 'San Francisco', 'Seattle']

}

Writing

df\_to\_write = pd.DataFrame(data)

# Write DataFrame to an Excel file

df\_to\_write.to\_excel("output.xlsx", index=False)

print("DataFrame has been written to 'output.xlsx'.")

WEEK-6

Merge

import pandas as pd

# Create two sample DataFrames

df1 = pd.DataFrame({

'ID': [1, 2, 3],

'Name': ['Alice', 'Bob', 'Charlie']

})

df2 = pd.DataFrame({

'ID': [2, 3, 4],

'Age': [25, 30, 22]

})

# Merge DataFrames based on the 'ID' column

merged\_df = pd.merge(df1, df2, on='ID', how='inner')

print("Merged DataFrame:")

print(merged\_df)

# Merge based on multiple columns

merged\_df = pd.merge(df1, df2, on=['ID'], how='inner')

Combining DataFrame

import pandas as pd

# Create two sample DataFrames

df1 = pd.DataFrame({

'ID': [1, 2, 3],

'Name': ['Alice', 'Bob', 'Charlie']

})

df2 = pd.DataFrame({

'ID': [4, 5, 6],

'Name': ['David', 'Eva', 'Frank']

})

# Concatenate DataFrames along rows

concatenated\_df = pd.concat([df1, df2], ignore\_index=True)

print("Concatenated DataFrame along rows:")

print(concatenated\_df)

# Create two sample DataFrames

df1 = pd.DataFrame({

'ID': [1, 2, 3],

'Name': ['Alice', 'Bob', 'Charlie']

})

df2 = pd.DataFrame({

'Age': [25, 30, 22],

'City': ['New York', 'San Francisco', 'Seattle']

})

# Concatenate DataFrames along columns

concatenated\_df = pd.concat([df1, df2], axis=1)

print("Concatenated DataFrame along columns:")

print(concatenated\_df)

Pivoting

import pandas as pd

# Create a sample DataFrame

data = {

'Date': ['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-02'],

'Category': ['A', 'B', 'A', 'B'],

'Sales': [100, 150, 200, 250]

}

df = pd.DataFrame(data)

# Display the original DataFrame

print("Original DataFrame:")

print(df)

# Pivot the DataFrame

pivot\_df = df.pivot(index='Date', columns='Category', values='Sales')

# Display the pivoted DataFrame

print("\nPivoted DataFrame:")

print(pivot\_df)

Duplicates

import pandas as pd

# Create a sample DataFrame with duplicate values in the 'Name' column

data = {

'ID': [1, 2, 3, 4, 5],

'Name': ['Alice', 'Bob', 'Charlie', 'Bob', 'Alice'],

'Age': [25, 30, 22, 30, 25]

}

df = pd.DataFrame(data)

# Display the original DataFrame

print("Original DataFrame:")

print(df)

# Identify duplicate rows based on the 'Name' column

duplicates = df.duplicated(subset='Name')

# Display the rows that are duplicates based on the 'Name' column

print("\nDuplicate Rows based on 'Name':")

print(df[duplicates])

# Drop duplicate rows based on the 'Name' column

df\_no\_duplicates = df.drop\_duplicates(subset='Name')

# Display the DataFrame without duplicates based on the 'Name' column

print("\nDataFrame without Duplicates based on 'Name':")

print(df\_no\_duplicates)

Mapping

import pandas as pd

# Create a sample DataFrame

data = {

'Numbers': [1, 2, 3, 4, 5]

}

df = pd.DataFrame(data)

# Display the original DataFrame

print("Original DataFrame:")

print(df)

# Define a function to square a number

def square(x):

return x \*\* 2

# Use the apply function to apply the square function to the 'Numbers' column

df['Squared'] = df['Numbers'].apply(square)

# Display the DataFrame with squared numbers

print("\nDataFrame with Squared Numbers:")

print(df)

df['Squared'] = df['Numbers'].apply(lambda x: x \*\* 2)

binning

import pandas as pd

# Create a sample DataFrame with numerical data

data = {

'Scores': [75, 82, 95, 68, 60, 90, 78, 88, 72, 85]

}

df = pd.DataFrame(data)

# Define bin edges and labels

bins = [0, 60, 70, 80, 90, 100]

labels = ['F', 'D', 'C', 'B', 'A']

# Create a new column 'Grade' by binning the 'Scores' column

df['Grade'] = pd.cut(df['Scores'], bins=bins, labels=labels, right=False)

# Display the original DataFrame and the DataFrame with the 'Grade' column

print("Original DataFrame:")

print(df)

Outlier

import pandas as pd

from scipy.stats import zscore

# Create a sample DataFrame with numerical data

data = {

'Values': [10, 15, 12, 18, 22, 8, 25, 30, 5, 35, 40]

}

df = pd.DataFrame(data)

# Calculate Z-scores for each value in the 'Values' column

z\_scores = zscore(df['Values'])

# Define a threshold for identifying outliers (e.g., Z-score greater than 3 or less than -3)

threshold = 3

# Identify outliers based on the threshold

outliers = (z\_scores > threshold) | (z\_scores < -threshold)

# Add a new column 'Is\_Outlier' to the DataFrame indicating whether each value is an outlier

df['Is\_Outlier'] = outliers

# Display the original DataFrame and the DataFrame with the 'Is\_Outlier' column

print("Original DataFrame:")

print(df)

**WEEK-7**

**Working with Data PART-III**

Group by on Data Frame

import pandas as pd

# Create a sample DataFrame

data = {

'Category': ['A', 'B', 'A', 'B', 'A', 'B'],

'Value': [10, 15, 20, 25, 30, 35]

}

df = pd.DataFrame(data)

# Group by 'Category' and calculate the mean for each group

grouped\_df = df.groupby('Category').mean()

# Display the original DataFrame and the result of the groupby operation

print("Original DataFrame:")

print(df)

print("\nGrouped DataFrame (mean for each category):")

print(grouped\_df)

Splitting Applying and Combining

Splitting: The groupby('Category') operation is used to split the DataFrame into groups based on the 'Category' column.

Applying: The ['Sales'].sum() operation is applied to each group to calculate the sum of sales for each category.

Combining: The result is a Pandas Series that represents the total sales for each category.

import pandas as pd

# Create a sample DataFrame

data = {

'Category': ['A', 'B', 'A', 'B', 'A', 'B'],

'Sales': [100, 150, 200, 120, 180, 220]

}

df = pd.DataFrame(data)

# Step 1: Splitting - Group by 'Category'

grouped\_df = df.groupby('Category')

# Step 2: Applying - Calculate the sum for each group

sum\_per\_category = grouped\_df['Sales'].sum()

# Step 3: Combining - Display the result

print("Total Sales per Category:")

print(sum\_per\_category)

# Calculate multiple aggregations at once

result = grouped\_df['Sales'].agg(['sum', 'mean', 'count'])

# Display the result

print("Aggregations per Category:")

print(result)

Cross Tabulation

pd.crosstab() function to perform cross-tabulation on a DataFrame. Cross-tabulation is a way to summarize and analyze the relationship between two or more categorical variables.

import pandas as pd

# Create a sample DataFrame

data = {

'Category': ['A', 'B', 'A', 'B', 'A', 'B'],

'Color': ['Red', 'Blue', 'Red', 'Green', 'Blue', 'Red']

}

df = pd.DataFrame(data)

# Perform cross-tabulation

cross\_tab = pd.crosstab(df['Category'], df['Color'])

# Display the cross-tabulation result

print("Cross-Tabulation:")

print(cross\_tab)

# Specify additional parameters for the crosstab function

cross\_tab = pd.crosstab(df['Category'], df['Color'], margins=True, margins\_name='Total')

# Display the modified cross-tabulation result

print("Modified Cross-Tabulation:")

print(cross\_tab)

WEEK-8 DATA VISUALIZATION

Installing Seaborn

pip install seaborn

import seaborn as sns

Histograms

pip install seaborn matplotlib

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = [5, 10, 15, 20, 25, 30, 30, 35, 40, 45, 50]

# Create a histogram using seaborn

sns.histplot(data, bins=10, kde=False, color='skyblue')

# Add labels and title

plt.xlabel('Values')

plt.ylabel('Frequency')

plt.title('Histogram Example')

# Show the plot

plt.show()

Combining Plot Styles

import seaborn as sns

import matplotlib.pyplot as plt

# Set the style to 'darkgrid'

sns.set(style='darkgrid')

# Sample data

data = sns.load\_dataset('tips')

# Create a figure with two subplots

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))

# Plot a histogram on the first subplot

sns.histplot(data['total\_bill'], bins=20, kde=True, color='skyblue', ax=axes[0])

axes[0].set\_title('Histogram')

# Plot a scatter plot on the second subplot

sns.scatterplot(x='total\_bill', y='tip', data=data, color='coral', ax=axes[1])

axes[1].set\_title('Scatter Plot')

# Adjust layout

plt.tight\_layout()

# Show the combined plot

plt.show()

WEEK-9

Box and Violin Plots

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = sns.load\_dataset('tips')

# Set the style to 'whitegrid'

sns.set(style='whitegrid')

# Create a box plot

plt.figure(figsize=(8, 6))

sns.boxplot(x='day', y='total\_bill', data=data, palette='Set2')

plt.title('Box Plot of Total Bill by Day')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = sns.load\_dataset('tips')

# Set the style to 'whitegrid'

sns.set(style='whitegrid')

# Create a violin plot

plt.figure(figsize=(8, 6))

sns.violinplot(x='day', y='total\_bill', data=data, palette='Pastel1')

plt.title('Violin Plot of Total Bill by Day')

plt.show()

regression plots

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = sns.load\_dataset('tips')

# Set the style to 'whitegrid'

sns.set(style='whitegrid')

# Create a box plot

plt.figure(figsize=(8, 6))

sns.boxplot(x='day', y='total\_bill', data=data, palette='Set2')

plt.title('Box Plot of Total Bill by Day')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = sns.load\_dataset('tips')

# Set the style to 'whitegrid'

sns.set(style='whitegrid')

# Create a violin plot

plt.figure(figsize=(8, 6))

sns.violinplot(x='day', y='total\_bill', data=data, palette='Pastel1')

plt.title('Violin Plot of Total Bill by Day')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = sns.load\_dataset('tips')

# Set the style to 'whitegrid'

sns.set(style='whitegrid')

# Create a scatter plot with a regression line using lmplot

sns.lmplot(x='total\_bill', y='tip', data=data, height=6, aspect=1.5, scatter\_kws={'s': 30}, line\_kws={'color': 'blue'})

plt.title('Scatter Plot with Regression Line')

plt.show()

heatmaps and clustered matrice

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = sns.load\_dataset('flights')

# Pivot the data to create a matrix

flights\_matrix = data.pivot\_table(index='month', columns='year', values='passengers')

# Set the style to 'whitegrid'

sns.set(style='whitegrid')

# Create a heatmap

plt.figure(figsize=(10, 8))

sns.heatmap(flights\_matrix, cmap='YlGnBu', annot=True, fmt='d', linewidths=.5)

plt.title('Flights Data - Heatmap')

plt.show()

import seaborn as sns

import matplotlib.pyplot as plt

# Sample data

data = sns.load\_dataset('flights')

# Pivot the data to create a matrix

flights\_matrix = data.pivot\_table(index='month', columns='year', values='passengers')

# Set the style to 'whitegrid'

sns.set(style='whitegrid')

# Create a clustered matrix

plt.figure(figsize=(10, 8))

sns.clustermap(flights\_matrix, cmap='YlGnBu', annot=True, fmt='d', linewidths=.5)

plt.title('Flights Data - Clustered Matrix')

plt.show()

WEEK-10

USE CASES

Stock market analysis ,customer segmentation , credit card fraud

detection

pip install yfinance pandas matplotlib

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

def stock\_analysis(ticker, start\_date, end\_date):

# Download historical stock data

stock\_data = yf.download(ticker, start=start\_date, end=end\_date)

# Calculate daily returns

stock\_data['Daily\_Return'] = stock\_data['Adj Close'].pct\_change()

# Calculate cumulative returns

stock\_data['Cumulative\_Return'] = (1 + stock\_data['Daily\_Return']).cumprod()

# Plotting

plt.figure(figsize=(10, 6))

# Plot stock prices

plt.subplot(2, 1, 1)

plt.plot(stock\_data['Adj Close'])

plt.title(f'{ticker} Stock Price')

plt.xlabel('Date')

plt.ylabel('Stock Price')

# Plot cumulative returns

plt.subplot(2, 1, 2)

plt.plot(stock\_data['Cumulative\_Return'], color='r')

plt.title(f'{ticker} Cumulative Returns')

plt.xlabel('Date')

plt.ylabel('Cumulative Returns')

plt.tight\_layout()

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

# Input stock symbol (ticker), start date, and end date

stock\_ticker = input("Enter stock symbol (e.g., AAPL): ")

start\_date = input("Enter start date (YYYY-MM-DD): ")

end\_date = input("Enter end date (YYYY-MM-DD): ")

# Perform stock analysis

stock\_analysis(stock\_ticker, start\_date, end\_date)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

pip install pandas numpy matplotlib scikit-learn

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Generate some random customer data for demonstration purposes

np.random.seed(42)

data = {

'Age': np.random.randint(18, 65, 100),

'Income': np.random.randint(20000, 100000, 100),

}

df = pd.DataFrame(data)

# Standardize the data

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(df)

# Determine the optimal number of clusters using the Elbow Method

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

kmeans.fit(scaled\_data)

wcss.append(kmeans.inertia\_)

# Plot the Elbow Method graph

plt.plot(range(1, 11), wcss)

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS') # Within-Cluster Sum of Squares

plt.show()

# Based on the Elbow Method, choose the optimal number of clusters (k)

k\_optimal = int(input("Enter the optimal number of clusters (k): "))

# Apply k-means clustering

kmeans = KMeans(n\_clusters=k\_optimal, init='k-means++', max\_iter=300, n\_init=10, random\_state=0)

df['Cluster'] = kmeans.fit\_predict(scaled\_data)

# Display the segmented customers

print("\nSegmented Customers:")

print(df.groupby('Cluster').mean())

# Visualize the clusters

plt.scatter(scaled\_data[:, 0], scaled\_data[:, 1], c=df['Cluster'], cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red')

plt.title('Customer Segmentation')

plt.xlabel('Scaled Age')

plt.ylabel('Scaled Income')

plt.show()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

pip install pandas scikit-learn

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import LocalOutlierFactor

from sklearn.metrics import classification\_report, confusion\_matrix

# Load the credit card fraud dataset (replace with your dataset)

df = pd.read\_csv('credit\_card\_fraud\_dataset.csv')

# Separate features and labels

X = df.drop('Class', axis=1)

y = df['Class']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Train the Local Outlier Factor model

model = LocalOutlierFactor(contamination=0.01)

y\_pred = model.fit\_predict(X\_test)

# Convert the predictions (-1 for anomalies, 1 for normal) to binary labels (0 for normal, 1 for fraud)

y\_pred\_binary = [1 if pred == -1 else 0 for pred in y\_pred]

# Evaluate the model

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_binary))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred\_binary))