SRM Institute of Science and Technology
Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204
Department of Computer Applications
Circular – 2023-24
BCA DS 3rd Sem

Data Wrangling (UDS23G01J)

Lab Manual

Here's a lab manual outline based on the list of programs. I'll provide a basic structure and Python code examples. You can expand on these with more detailed explanations and specific datasets.

Lab 1: Install Python, Setting up IPython and Jupyter Notebook

Title: Python and Environment Setup

Aim: To install Python, set up the IPython interactive shell, and install and configure the Jupyter Notebook environment.

Procedure:

1. Download Python from the official website (python.org).
2. Install Python, ensuring to add it to your system's PATH during installation.
3. Open a terminal or command prompt and check the Python installation by typing python --version.
4. Install IPython using pip: pip install ipython.
5. Install Jupyter Notebook using pip: pip install notebook.
6. Launch Jupyter Notebook by typing jupyter notebook in the terminal.

Source Code:

7. No direct source code is involved in this lab, as it mainly focuses on installation and setup.

Input:

8. No input is required for this lab.

Expected Output:

9. Successful installation of Python, IPython, and Jupyter Notebook.
10. Ability to open and run a Jupyter Notebook in a web browser.

Lab 2: Using Python Libraries to Parse Excel File

Title: Parsing Excel Files with Python

Aim: To use the pandas library to parse and read data from an Excel file.

Procedure:

1. Install the pandas and openpyxl (for .xlsx files) libraries: pip install pandas openpyxl
2. Import the pandas library in your Python script or Jupyter Notebook.
3. Use the pd.read_excel() function to read the Excel file.
4. Access and manipulate the data using the DataFrame object.

Source Code:

```
import pandas as pd

# Read the Excel file
excel_file = 'data.xlsx'  # Replace with your file name
df = pd.read_excel(excel_file)

# Print the first 5 rows
print(df.head())

# Print the column names
print(df.columns)

# Get some info about the data
print(df.info())
```

Input:

5. An Excel file (data.xlsx in the example).

Expected Output:

6. The program should read the Excel file and display the first few rows, column names, and data types.

Lab 3: Using NumPy & Pandas to Calculate Basic Descriptive Statistics on the DataFrame

Title: Descriptive Statistics with NumPy and Pandas

Aim: To use NumPy and Pandas to calculate descriptive statistics (mean, median, standard deviation, etc.) on data in a DataFrame.

Procedure:

1. Import the NumPy and Pandas libraries.
2. Read data from a file (e.g., CSV or Excel) into a Pandas DataFrame.
3. Use Pandas functions like df.describe(), df.mean(), df.median(), and df.std() to calculate statistics. You can also use NumPy functions on Pandas Series (columns of the DataFrame).

Source Code:

```
import pandas as pd
import numpy as np

# Load data (replace with your data file)
data = {'col1': [1, 2, 3, 4, 5], 'col2': [6, 7, 8, 9, 10]}
df = pd.DataFrame(data)

# Calculate descriptive statistics using Pandas
print(df.describe())

# Calculate mean of a column using Pandas
print("Mean of col1:", df['col1'].mean())

# Calculate standard deviation of a column using NumPy
print("Standard deviation of col2:", np.std(df['col2']))
```

Input:

4. A data file (CSV, Excel, etc.).

Expected Output:

5. The program should output descriptive statistics for the numerical columns in the DataFrame.

Lab 4: Download a Dataset and Perform Visual Exploration of Data

Title: Data Visualization

Aim: To download a dataset and use the matplotlib library to perform visual exploration of the data.

Procedure:

1. Download a dataset (e.g., from Kaggle or the UCI Machine Learning Repository).
2. Import the pandas and matplotlib.pyplot libraries.
3. Read the data into a Pandas DataFrame.
4. Use matplotlib.pyplot functions like plt.scatter(), plt.hist(), and plt.boxplot() to create visualizations.

Source Code:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
data = {'col1': [1, 2, 3, 4, 5], 'col2': [6, 7, 8, 9, 10], 'col3':
[2,4,6,8,10]} #Example Data
df = pd.DataFrame(data)

# Create a scatter plot
plt.scatter(df['col1'], df['col2'])
plt.xlabel('Column 1')
plt.ylabel('Column 2')
plt.title('Scatter Plot of Col1 vs Col2')
plt.show()

# Create a histogram
plt.hist(df['col1'], bins=5)
plt.xlabel('Column 1')
plt.ylabel('Frequency')
plt.title('Histogram of Col1')
plt.show()
```

Input:

5. A dataset (e.g., CSV file).

Expected Output:

6. The program should generate visual plots (scatter plots, histograms, box plots, etc.) of the data.

Lab 5: Use RegEx for Text Format Files

Title: Regular Expressions for Text Processing

Aim: To use regular expressions (RegEx) to extract and manipulate text data from files.

Procedure:

1. Import the re module in Python.
2. Read the text data from a file.
3. Define regular expression patterns to match specific text.
4. Use re.search(), re.findall(), re.sub(), etc., to find, extract, and replace text based on the patterns.

Source Code:

```
import re

# Sample text data
text = "Name: John Doe, Age: 30, Email: john.doe@example.com"

# Define a regular expression pattern to extract the email address
email_pattern = r"[\w\.-]+@[\w\.-]+"

# Find the email address
match = re.search(email_pattern, text)
if match:
    print("Email:", match.group())

# Find all occurrences of numbers
age_pattern = r"\d+"
ages = re.findall(age_pattern, text)
print("Ages:", ages)

# Replace "John" with "Jane"
new_text = re.sub(r"John", "Jane", text)
print("Modified text:", new_text)
```

Input:

5. A text file or string containing the data to be processed.

Expected Output:

6. The program should extract, find, or modify text data according to the specified regular expression patterns.

Lab 6: Build a Web Scrapper using Python

Title: Web Scraping with Python

Aim: To build a web scraper using Python to extract data from a website.

Procedure:

1. Install the requests and BeautifulSoup4 libraries: pip install requests beautifulsoup4
2. Import the necessary libraries.
3. Use the requests library to fetch the HTML content of the web page.
4. Use BeautifulSoup4 to parse the HTML and navigate the document structure.
5. Locate the desired data using HTML tags and attributes.
6. Extract the data and store it in a suitable format (e.g., a list or DataFrame).

Source Code:

```
import requests
from bs4 import BeautifulSoup

# URL of the website to scrape
url = "https://example.com"  # Replace with a real URL

# Fetch the HTML content
response = requests.get(url)
html_content = response.text

# Parse the HTML using BeautifulSoup
soup = BeautifulSoup(html_content, 'html.parser')

# Example: Extract all the links from the page
links = []
for a_tag in soup.find_all('a'):
    link = a_tag.get('href')
    if link: # Check if link is not None
      links.append(link)

print("Links found:", links)

# Example: Extract all the titles
titles = [title.text for title in soup.find_all('title')]
print("Titles:", titles)
```

Input:

7. A URL of the website to scrape.

Expected Output:

8. The program should extract the desired data from the website (e.g., links, titles, paragraphs, tables).

Lab 7: Explore different data cleaning Tools

Title: Data Cleaning Tools

Aim: To explore various Python tools and techniques for cleaning data.

Procedure:

1. Import the pandas library.
2. Load a dataset that requires cleaning.
3. Use Pandas functions to identify and handle:
    - Missing values: df.isnull(), df.fillna(), df.dropna()
    - Duplicate values: df.duplicated(), df.drop_duplicates()
    - Incorrect data types: df.dtypes, df.astype()
    - Inconsistent formatting: String manipulation functions
    - Outliers: (Covered in Lab 8, but introduce the concept here)

Source Code:

```
import pandas as pd
import numpy as np

# Sample data with issues
data = {'col1': [1, 2, None, 4, 5, 2],
        'col2': ['A', 'B', 'A', 'C', 'B', 'A'],
        'col3': [10, 20, 30, 40, 50, 20],
        'col4': [100, 200, 100, 200, 300, 100]}
df = pd.DataFrame(data)

# Handle missing values by filling with the mean of 'col1'
df['col1'].fillna(df['col1'].mean(), inplace=True)
print("After filling missing values:\n", df)

# Handle duplicate rows
df.drop_duplicates(inplace=True)
print("\nAfter removing duplicates:\n", df)

# Correct data type (if needed, col2 is fine as is)
df['col1'] = df['col1'].astype(int)

print("\nAfter type conversion:\n", df)
```

Input:

4. A dataset with missing values, duplicates, and/or inconsistent data.

Expected Output:

5. The program should output the cleaned DataFrame.

Lab 8: Outlier Detection Using a Simple Statistical Test

Title: Outlier Detection

Aim: To detect outliers in a dataset using a statistical test (e.g., the z-score method).

Procedure:

1. Import the pandas and scipy.stats libraries.
2. Load the dataset.
3. Calculate the z-scores for the relevant column(s).
4. Define a threshold for the z-score (e.g., 3).
5. Identify data points with z-scores exceeding the threshold as outliers.

Source Code:

```
import pandas as pd
from scipy import stats

# Sample data
data = {'col1': [1, 2, 3, 4, 5, 100]}
df = pd.DataFrame(data)

# Calculate z-scores for 'col1'
z_scores = stats.zscore(df['col1'])
print("Z-Scores", z_scores)
# Define the threshold
threshold = 3

# Identify outliers
outlier_indices = df['col1'][abs(z_scores) > threshold].index
outliers = df.loc[outlier_indices]
print("Outliers:", outliers)

# Create a new column indicating outliers
df['is_outlier'] = abs(z_scores) > threshold
print(df)
```

Input:

6. A dataset with potential outliers.

Expected Output:

7. The program should identify and output the outlier data points.

Lab 9: Read any Tabular Dataset and Perform Data Cleaning

Title: Data Cleaning Pipeline

Aim: To read a tabular dataset and perform a comprehensive data cleaning process.

Procedure:

1. Import the pandas library.
2. Read the tabular dataset (CSV, Excel, etc.) into a DataFrame.
3. Implement a series of data cleaning steps:

    Handle missing values.

    Remove duplicate rows.

    Correct data types.

    Handle inconsistent formatting.

    Detect and handle outliers (using a method like z-score).

4. Print the cleaned DataFrame.

Source Code:

```
import pandas as pd
from scipy import stats

def clean_data(df):
    """
    Cleans the input DataFrame.

    Args:
        df: The Pandas DataFrame to clean.

    Returns:
        The cleaned Pandas DataFrame.
    """
    # Handle missing values (fill with mean)
    for col in df.columns:
        if df[col].dtype in ['int64', 'float64']:  # Only for numeric columns
            df[col].fillna(df[col].mean(), inplace=True)
        else:
            df[col].fillna(df[col].mode()[0], inplace=True) # Fill with mode
for non-numeric

    # Remove duplicate rows
    df.drop_duplicates(inplace=True)

    # Correct data types (example: convert a column to datetime, if
applicable)
    # df['date_column'] = pd.to_datetime(df['date_column'], errors='coerce')

    # Handle outliers (using z-score)
    for col in df.columns:
        if df[col].dtype in ['int64', 'float64']:
            z_scores = stats.zscore(df[col])
            df = df[abs(z_scores) <= 3]  # Remove rows with z-score > 3
```

```
    return df

# Load the dataset
data = {'col1': [1, 2, None, 4, 5, 2, 100],
        'col2': ['A', 'B', 'A', 'C', 'B', 'A', 'D'],
        'col3': [10, 20, 30, 40, 50, 20, 110],
        'col4': [100, 200, 100, 200, 300, 100, 400]}
df = pd.DataFrame(data)

# Clean the data
cleaned_df = clean_data(df.copy()) # Pass a copy to avoid modifying original
print("Cleaned Data:\n", cleaned_df)
```

Input:

5. A tabular dataset (CSV, Excel, etc.) with potential data quality issues.

Expected Output:

6. The program should output a cleaned version of the input dataset.

Lab 10: Implement Combine and Merge of Data in Pandas Object

Title: Combining Data with Pandas

Aim: To implement combining and merging data using Pandas.

Procedure:

1. Import the pandas library.
2. Create two or more DataFrames to combine.
3. Use pd.concat() to concatenate DataFrames along rows or columns.
4. Use pd.merge() to merge DataFrames based on common columns (like SQL joins).

Source Code:

```
import pandas as pd

# Create sample DataFrames
df1 = pd.DataFrame({'key': ['A', 'B', 'C', 'D'],
                    'value1': [1, 2, 3, 4]})
df2 = pd.DataFrame({'key': ['B', 'D', 'E', 'F'],
                    'value2': [5, 6, 7, 8]})
df3 = pd.DataFrame({'key2': ['B', 'D', 'E', 'F'],
                    'value3': [5, 6, 7, 8]})

# Concatenate along rows
result_concat_row = pd.concat([df1, df2], ignore_index=True)
print("Concatenate along rows:\n", result_concat_row)

# Concatenate along columns
result_concat_col = pd.concat([df1, df2], axis=1)
print("\nConcatenate along columns:\n", result_concat_col)

# Merge DataFrames on the 'key' column (inner join)
result_merge_inner = pd.merge(df1, df2, on='key', how='inner')
print("\nMerge (inner join):\n", result_merge_inner)

 # Merge DataFrames on different column names
result_merge_left = pd.merge(df1, df3, left_on='key', right_on='key2',
how='left')
print("\nMerge with different column names:\n", result_merge_left)

# Perform a left join
result_merge_left = pd.merge(df1, df2, on='key', how='left')
print("\nMerge (left join):\n", result_merge_left)

# Perform a right join
result_merge_right = pd.merge(df1, df2, on='key', how='right')
print("\nMerge (right join):\n", result_merge_right)

# Perform an outer join
result_merge_outer = pd.merge(df1, df2, on='key', how='outer')
print("\nMerge (outer join):\n", result_merge_outer)
```

Input:

5. Two or more DataFrames.

Expected Output:

6. The program should output the combined/merged DataFrames, demonstrating different concatenation and merging methods.

Lab 11: Implement Reshaping and Pivoting using Pandas Object

Title: Reshaping and Pivoting Data with Pandas

Aim: To implement reshaping and pivoting operations using Pandas.

Procedure:

1. Import the pandas library.
2. Create a DataFrame to reshape and pivot.
3. Use df.melt() to unpivot a DataFrame from wide to long format.
4. Use df.pivot_table() to create pivot tables and reshape data.

Source Code:

```
import pandas as pd

# Sample DataFrame
data = {'date': ['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-02'],
        'city': ['New York', 'Los Angeles', 'New York', 'Los Angeles'],
        'temperature': [32, 65, 35, 70],
        'humidity': [60, 40, 65, 35]}
df = pd.DataFrame(data)

# Melt the DataFrame
df_melted = pd.melt(df, id_vars=['date', 'city'], value_vars=['temperature',
'humidity'], var_name='variable', value_name='value')
print("Melted DataFrame:\n", df_melted)

# Create a pivot table
df_pivot = pd.pivot_table(df, values=['temperature', 'humidity'],
index='date', columns='city', aggfunc='mean')
print("\nPivot Table:\n", df_pivot)

# Create a more complex pivot table
df_pivot_complex = pd.pivot_table(df, values='temperature', index=['date',
'city'], aggfunc='mean')
print("\nComplex Pivot Table:\n", df_pivot_complex)
```

Input:

5. A DataFrame to reshape and pivot.

Expected Output:

6. The program should output the reshaped (melted) and pivoted DataFrames.

Lab 12: Use Matplotlib to Perform Data Visualization

Title: Data Visualization with Matplotlib

Aim: To use the matplotlib library to create various data visualizations.

Procedure:

1. Import the matplotlib.pyplot library.
2. Load or create data for visualization.
3. Use plt.plot() for line plots, plt.scatter() for scatter plots, plt.bar() for bar charts, plt.hist() for histograms, plt.boxplot() for box plots, and other matplotlib.pyplot functions.
4. Customize plots with labels, titles, legends, and styles.

Source Code:

```
import matplotlib.pyplot as plt
import numpy as np

# Sample data
x = np.array([1, 2, 3, 4, 5])
y1 = np.array([2, 4, 6, 8, 10])
y2 = np.array([1, 3, 5, 7, 9])

# Line plot
plt.plot(x, y1, label='Line 1', marker='o')
plt.plot(x, y2, label='Line 2', marker='x')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot Example')
plt.legend()
plt.show()

# Scatter plot
plt.scatter(x, y1)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot Example')
plt.show()

# Bar chart
categories = ['A', 'B', 'C', 'D']
values = [10, 15, 7, 12]
plt.bar(categories, values)
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart Example')
plt.show()

# Histogram
data = np.random.normal(0, 1, 100)  # Generate random data
plt.hist(data, bins=20)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram Example')
plt.show()

# Box plot
data = [np.random.normal(0, 1, 50), np.random.normal(2, 1, 50)]
plt.boxplot(data)
```

```
plt.ylabel('Value')
plt.title('Box Plot Example')
plt.show()
```

Input:

5. Data to be visualized (arrays, lists, DataFrames).

Expected Output:

6. The program should generate various types of plots (line plots, scatter plots, bar charts, histograms, box plots, etc.).

Lab 13: Perform Groupby Operations using Pandas

Title: Grouping Data with Pandas

Aim: To use Pandas to group data and perform calculations on the groups.

Procedure:

1. Import the Pandas library.
2. Create a DataFrame.
3. Use the df.groupby() method to group the DataFrame by one or more columns.
4. Apply aggregation functions (e.g., sum(), mean(), count(), min(), max()) to the grouped data.
5. Iterate through the groups and display the results.

Source Code:

```
import pandas as pd

# Create a sample DataFrame
data = {
    'date': ['2024-01-01', '2024-01-01', '2024-01-02', '2024-01-02', '2024-
01-03', '2024-01-03'],
    'category': ['A', 'B', 'A', 'B', 'A', 'B'],
    'value1': [10, 20, 15, 25, 12, 18],
    'value2': [30, 40, 35, 45, 28, 32]
}
df = pd.DataFrame(data)

# Group by 'category' and calculate the mean of 'value1' and 'value2'
grouped_mean = df.groupby('category')[['value1', 'value2']].mean()
print("Grouped mean:\n", grouped_mean)

# Group by 'date' and calculate the sum, count, and min of 'value1'
grouped_agg = df.groupby('date')['value1'].agg(['sum', 'count', 'min'])
print("\nGrouped aggregation:\n", grouped_agg)

# Group by multiple columns ('date' and 'category')
grouped_multiple = df.groupby(['date', 'category'])[['value1',
'value2']].sum()
print("\nGrouped by multiple columns:\n", grouped_multiple)

# Iterate through groups
for name, group in df.groupby('category'):
    print(f"\nGroup: {name}")
    print(group)
```

Input:

6. A DataFrame.

Expected Output:

7. The program should output the results of the groupby operations, including aggregated data for each group.

Lab 14: Perform Aggregation Operation on DataFrame

Title: Aggregating Data with Pandas

Aim: To use Pandas to perform aggregation operations on DataFrames.

Procedure:

1. Import the Pandas library.
2. Create a DataFrame.
3. Use the df.agg() method to apply one or more aggregation functions to the entire DataFrame or specific columns.
4. Use groupby() in conjunction with agg() for group-wise aggregation.

Source Code:

```
import pandas as pd
import numpy as np

# Create a sample DataFrame
data = {
    'A': [1, 2, 3, 4, 5],
    'B': [10, 20, 30, 40, 50],
    'C': [100, 200, 300, 400, 500],
    'D': ['a', 'b', 'a', 'b', 'a']
}
df = pd.DataFrame(data)

# Aggregate the entire DataFrame
aggregated_all = df.agg(['sum', 'mean', 'max', 'min'])
print("Aggregated all:\n", aggregated_all)

# Aggregate specific columns
aggregated_specific = df.agg({
    'A': ['sum', 'mean'],
    'B': ['max', 'min'],
    'C': 'median'  # Single function for column C
})
print("\nAggregated specific columns:\n", aggregated_specific)

# Group-wise aggregation
grouped_agg = df.groupby('D').agg({
    'A': 'mean',
    'B': 'sum',
    'C': ['min', 'max']
})
print("\nGroup-wise aggregation:\n", grouped_agg)
```

Input:

5. A DataFrame.

Expected Output:

6. The program should output the results of the aggregation operations, including aggregated values for the entire DataFrame or specific columns, and group-wise aggregations.

Lab 15: Perform Cross Tab Analysis in Python

Title: Cross-Tabulation with Pandas

Aim: To perform cross-tabulation analysis using Pandas.

Procedure:

1. Import the Pandas library.
2. Create a DataFrame with categorical data.
3. Use the pd.crosstab() function to create a cross-tabulation (contingency table) of two or more categorical variables.
4. Optionally, normalize the cross-tabulation to show proportions or percentages.

Source Code:

```
import pandas as pd

# Create a sample DataFrame
data = {
    'gender': ['Male', 'Female', 'Male', 'Female', 'Male', 'Female', 'Male',
'Female'],
    'smoker': ['Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes'],
    'age_group': ['Young', 'Young', 'Adult', 'Adult', 'Young', 'Adult',
'Young', 'Adult'],
    'count': [10,12,14,23,5,6,7,8]
}
df = pd.DataFrame(data)

# Create a simple cross-tabulation
cross_tab = pd.crosstab(df['gender'], df['smoker'])
print("Simple cross-tabulation:\n", cross_tab)

# Cross-tabulation with multiple variables
cross_tab_multi = pd.crosstab(df['gender'], [df['smoker'], df['age_group']])
print("\nCross-tabulation with multiple variables:\n", cross_tab_multi)

# Normalized cross-tabulation (row-wise)
cross_tab_norm_row = pd.crosstab(df['gender'], df['smoker'],
normalize='index')
print("\nNormalized cross-tabulation (row-wise):\n", cross_tab_norm_row)

 # Normalized cross-tabulation (column-wise)
cross_tab_norm_col = pd.crosstab(df['gender'], df['smoker'],
normalize='columns')
print("\nNormalized cross-tabulation (column-wise):\n", cross_tab_norm_col)

# Cross-tabulation with aggregation
cross_tab_agg = pd.crosstab(df['gender'], df['smoker'], values=df['count'],
aggfunc='sum')
print("\nCross-tabulation with aggregation:\n", cross_tab_agg)
```

Input:

5. A DataFrame with categorical data.

Expected Output:

6. The program should output cross-tabulation tables, optionally normalized, showing the relationship between the categorical variables.