SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA GAI 1st semester

Foundations of Data Science (PGI20D01J)

Lab Manual

Lab 1: Perform Analysis on Simple Dataset I for Data Science and Business Intelligence Applications

Title: Basic Data Analysis on a Simple Dataset

Aim: To introduce fundamental data analysis techniques using Python, including loading data, performing descriptive statistics, and generating basic visualizations for business intelligence applications.

Procedure:

1. **Identify a Simple Dataset:** Choose a readily available and simple dataset (e.g., a CSV file with a few columns and rows, like sales data, student scores, or weather data).
2. **Load the Dataset:** Use the Pandas library to load the dataset into a DataFrame.
3. **Explore Data Types and Structure:** Check the data types of columns and the overall shape of the DataFrame.
4. **Descriptive Statistics:** Calculate basic descriptive statistics (mean, median, mode, standard deviation, min, max, quartiles) for numerical columns.
5. **Frequency Counts:** Determine the frequency of unique values for categorical columns.
6. **Basic Visualization:** Create simple plots such as histograms for numerical data and bar charts for categorical data to understand data distribution.
7. **Interpret Results:** Analyze the statistics and visualizations to draw initial insights from the dataset.

Source Code:

```
# Lab 1: Basic Data Analysis on a Simple Dataset

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# --- Configuration ---
# Define the path to your dataset. Replace 'your_dataset.csv' with the actual
file name.
DATASET_PATH = 'your_dataset.csv'

# --- Procedure ---

try:
    # 1. Load the Dataset
    # Assuming the dataset is a CSV file. Adjust read_csv for other formats
(e.g., read_excel, read_json).
```

```
    df = pd.read_csv(DATASET_PATH)
    print(f"Successfully loaded dataset from: {DATASET_PATH}\n")

    # 2. Explore Data Types and Structure
    print("--- Dataset Information ---")
    df.info()
    print("\nFirst 5 rows of the dataset:")
    print(df.head())
    print(f"\nDataset shape (rows, columns): {df.shape}")

    # 3. Descriptive Statistics for Numerical Columns
    print("\n--- Descriptive Statistics for Numerical Columns ---")
    print(df.describe())

    # 4. Frequency Counts for Categorical Columns (Example: assuming 'Category'
is a categorical column)
    # You might need to identify your categorical columns and iterate through
them.
    print("\n--- Frequency Counts for Categorical Columns (Example) ---")
    for col in df.select_dtypes(include='object').columns: # Selects columns
with object (string) dtype
        print(f"\nFrequency counts for '{col}':")
        print(df[col].value_counts())

    # 5. Basic Visualization
    print("\n--- Generating Basic Visualizations ---")

    # Set a style for plots
    sns.set_style("whitegrid")

    # Example: Histogram for a numerical column (e.g., 'Value' or 'Sales')
    # Replace 'Numerical_Column' with an actual numerical column from your
dataset
    numerical_cols = df.select_dtypes(include=['number']).columns
    if not numerical_cols.empty:
        for num_col in numerical_cols:
            plt.figure(figsize=(8, 5))
            sns.histplot(df[num_col], kde=True)
            plt.title(f'Distribution of {num_col}')
            plt.xlabel(num_col)
            plt.ylabel('Frequency')
            plt.show()
    else:
        print("No numerical columns found for histogram visualization.")

    # Example: Bar chart for a categorical column (e.g., 'Product_Category')
    # Replace 'Categorical_Column' with an actual categorical column from your
dataset
    categorical_cols = df.select_dtypes(include='object').columns
    if not categorical_cols.empty:
        for cat_col in categorical_cols:
            plt.figure(figsize=(10, 6))
            df[cat_col].value_counts().plot(kind='bar')
            plt.title(f'Frequency of {cat_col}')
            plt.xlabel(cat_col)
            plt.ylabel('Count')
            plt.xticks(rotation=45, ha='right')
            plt.tight_layout()
            plt.show()
    else:
        print("No categorical columns found for bar chart visualization.")

    print("\n--- Analysis Complete ---")

except FileNotFoundError:
    print(f"Error: The file '{DATASET_PATH}' was not found. Please ensure the
dataset is in the correct directory.")
```

```
except Exception as e:
    print(f"An error occurred: {e}")
```

Input: A CSV file named `your_dataset.csv` (or similar, depending on your modification in the source code) placed in the same directory as your Python script. **Example `your_dataset.csv` content:**

```
ID,Product,Category,Price,Quantity,Region
1,Laptop,Electronics,1200,10,North
2,Mouse,Electronics,25,50,South
3,Keyboard,Electronics,75,30,North
4,Desk,Furniture,300,5,East
5,Chair,Furniture,150,15,West
6,Monitor,Electronics,250,20,South
7,Table,Furniture,200,8,East
8,Headphones,Electronics,100,40,North
9,Lamp,Furniture,50,25,West
10,Webcam,Electronics,60,35,South
```

Expected Output: The output will include:

- Confirmation of successful dataset loading.
- Dataset information (`df.info()`) showing column names, non-null counts, and data types.
- The first few rows of the DataFrame (`df.head()`).
- Descriptive statistics for numerical columns (`df.describe()`).
- Frequency counts for categorical columns.
- Several plots displayed: histograms for numerical columns and bar charts for categorical columns, showing data distributions.
- A final message indicating "Analysis Complete."

Lab 2: Perform Analysis on Simple Dataset II for Data Science and Business Intelligence Applications

Title: Advanced Data Analysis and Preprocessing on a Simple Dataset

Aim: To apply more advanced data analysis techniques, including data cleaning, handling missing values, and simple feature engineering, to prepare a dataset for further analysis or modeling.

Procedure:

1. **Load the Dataset:** Load the same or a slightly more complex dataset as in Lab 1.
2. **Identify Missing Values:** Check for missing values in the dataset and quantify them.
3. **Handle Missing Values:** Implement strategies to handle missing values (e.g., imputation with mean/median/mode, or dropping rows/columns).
4. **Identify and Handle Duplicates:** Check for and remove duplicate rows if present.
5. **Data Type Conversion:** Convert columns to appropriate data types if necessary (e.g., converting a string column to numeric).
6. **Simple Feature Engineering:** Create new features from existing ones (e.g., combining columns, extracting information from date/time columns).
7. **Data Transformation (Optional):** Apply simple transformations like log transformation or standardization/normalization to numerical features if applicable.
8. **Verify Changes:** Review the DataFrame after transformations to ensure changes are applied correctly.

Source Code: *(Provide Python code using Pandas for data cleaning, handling missing values, and feature engineering. Example functions like* `df.isnull().sum()`, `df.fillna()`, `df.drop_duplicates()`, `pd.to_numeric()`, *and creating new columns based on existing ones.)*

Input: A CSV file (potentially with missing values or inconsistencies) similar to Lab 1's input, but perhaps with some intentional missing data points or duplicates.

Expected Output:

- Output showing the count of missing values before and after handling.
- Confirmation of duplicate rows removed.
- Updated `df.info()` showing corrected data types.
- The first few rows of the transformed DataFrame, showcasing new features.

Lab 3: Collect and understand a simple data for a Data Science Application

Title: Data Collection and Initial Understanding

Aim: To understand various methods of data collection and perform an initial assessment of the collected data's structure, quality, and relevance for a data science application.

Procedure:

1. **Identify Data Source:** Determine a simple data source (e.g., a public CSV file, a simple web page for scraping, or a free API endpoint).
2. **Data Collection Method:**
    - **For existing files:** Download and load the file.
    - **For web scraping:** Use libraries like `requests` and `BeautifulSoup` to extract data from a simple HTML table or list.
    - **For API:** Use `requests` to make API calls and parse JSON responses.
3. **Initial Data Exploration:**
    - Examine the raw data structure.
    - Identify data types of collected fields.
    - Note any immediate quality issues (e.g., inconsistent formatting, missing values, irrelevant information).
4. **Document Data Characteristics:** Summarize the source, collection method, and initial observations about the data.

Source Code: *(Provide Python code for either loading a file, a basic web scraping example, or a simple API call using `requests` and `json`.)*

Input:

- **For file:** A URL to a public CSV/JSON file or a local file path.
- **For web scraping:** A URL of a simple webpage with data.
- **For API:** A base URL and example endpoint for a public API.

Expected Output:

- The raw collected data (e.g., printed DataFrame head, JSON response).
- A summary of data characteristics, including identified data types and initial quality notes.

Lab 4: Perform Analysis on Simple Data for Mathematical, Numerical, Data Engineering Processing

Title: Mathematical and Numerical Processing for Data Engineering

Aim: To apply mathematical and numerical operations on simple datasets, focusing on techniques relevant for data engineering and preparing data for analytical models.

Procedure:

1. **Load Data:** Load a numerical dataset.
2. **Mathematical Operations:** Perform element-wise operations (addition, subtraction, multiplication, division), aggregation (sum, average, min, max), and statistical functions (variance, standard deviation).
3. **Numerical Transformations:** Apply transformations like logarithms, exponentials, square roots.
4. **Vectorization:** Utilize NumPy for efficient vectorized operations on arrays.
5. **Handling Outliers (Basic):** Implement simple methods to identify and potentially cap/floor outliers.
6. **Data Reshaping:** Practice reshaping data (e.g., pivot tables, stacking/unstacking) to prepare it for different analytical views.

Source Code: *(Provide Python code using Pandas and NumPy for mathematical operations, numerical transformations, and data reshaping.)*

Input: A numerical dataset, possibly with some outliers.

Expected Output:

- Results of various mathematical and numerical operations.
- Transformed data.
- Reshaped dataframes/arrays.

Lab 5: Install Python and apply all basic python functions

Title: Python Installation and Basic Functions

Aim: To successfully install Python and practice fundamental Python programming concepts, including data types, operators, control flow, functions, and basic data structures.

Procedure:

1. **Python Installation:** Install Python (e.g., via Anaconda or directly from python.org).
2. **IDE/Editor Setup:** Set up a suitable Integrated Development Environment (IDE) or text editor (e.g., VS Code, PyCharm, Jupyter Notebook).
3. **Basic Syntax:** Write and execute simple Python scripts.
4. **Data Types:** Experiment with `int`, `float`, `str`, `bool`, `list`, `tuple`, `dict`, `set`.
5. **Operators:** Practice arithmetic, comparison, logical, and assignment operators.
6. **Control Flow:** Implement `if-elif-else` statements, `for` loops, and `while` loops.
7. **Functions:** Define and call custom functions with parameters and return values.
8. **Input/Output:** Use `input()` for user input and `print()` for output.

Source Code: *(Provide a Python script demonstrating all the basic concepts mentioned in the procedure.)*

Input: User input for some interactive examples (e.g., asking for a number, a string).

Expected Output:

- Output from various `print()` statements demonstrating data types, operator results, control flow execution paths, and function outputs.
- Interactive prompts and responses.

Lab 6: Install and perform a Numerical Array Processing using NumPy

Title: Numerical Array Processing with NumPy

Aim: To install NumPy and perform efficient numerical array operations, which are foundational for scientific computing and data analysis in Python.

Procedure:

1. **NumPy Installation:** Install the NumPy library.
2. **Array Creation:** Create NumPy arrays from lists, using `np.zeros()`, `np.ones()`, `np.arange()`, `np.linspace()`.
3. **Array Attributes:** Explore array attributes like `shape`, `ndim`, `dtype`, `size`.
4. **Indexing and Slicing:** Practice accessing and modifying array elements using indexing and slicing.
5. **Array Operations:** Perform element-wise arithmetic operations, broadcasting, and matrix operations (dot product).
6. **Aggregation Functions:** Use `np.sum()`, `np.mean()`, `np.std()`, `np.min()`, `np.max()`.
7. **Reshaping Arrays:** Reshape arrays using `reshape()` and `flatten()`.

Source Code: *(Provide Python code demonstrating NumPy array creation, manipulation, operations, and aggregation.)*

Input: No specific user input, operations will be on predefined arrays.

Expected Output:

- Printed arrays at various stages of creation and transformation.
- Results of array operations and aggregation functions.
- Array attributes.

Lab 7: Apply Scientific functions on a given dataset with SciPy

Title: Applying Scientific Functions with SciPy

Aim: To install SciPy and apply its scientific computing functions to a given dataset, focusing on areas like optimization, interpolation, signal processing, and statistics.

Procedure:

1. **SciPy Installation:** Install the SciPy library.
2. **Dataset Preparation:** Use a simple numerical dataset (e.g., a NumPy array or Pandas Series).
3. **Statistical Functions:** Apply statistical tests (e.g., t-test, chi-squared test) or distributions from `scipy.stats`.
4. **Optimization (Basic):** Use `scipy.optimize` for a simple optimization problem (e.g., finding the minimum of a function).
5. **Interpolation (Basic):** Perform linear or cubic interpolation on a set of data points using `scipy.interpolate`.
6. **Signal Processing (Basic):** Apply a simple filter or transformation from `scipy.signal` (e.g., convolution).

Source Code: *(Provide Python code using SciPy modules like `scipy.stats`, `scipy.optimize`, `scipy.interpolate`, `scipy.signal`.)*

Input: A numerical dataset (e.g., a NumPy array or Pandas Series).

Expected Output:

- Results from statistical tests.
- Optimized function values or parameters.
- Interpolated data points.
- Transformed signal data.

Lab 8: Install, Import Pandas Learn and Explore a Sample Dataset with it

Title: Introduction to Pandas for Data Exploration

Aim: To install and import the Pandas library, and to learn its fundamental functionalities for loading, manipulating, and exploring a sample dataset.

Procedure:

1. **Pandas Installation:** Install the Pandas library.
2. **Import Pandas:** Import Pandas in your Python script.
3. **Load Dataset:** Load a sample dataset (e.g., CSV, Excel) into a Pandas DataFrame.
4. **Basic Exploration:**
    - View the first/last few rows (`.head()`, `.tail()`).
    - Get summary information (`.info()`).
    - Describe numerical columns (`.describe()`).
    - Check for missing values (`.isnull().sum()`).
    - Get unique values and their counts (`.value_counts()`).
5. **Selection and Filtering:** Select columns, rows, and filter data based on conditions.
6. **Sorting:** Sort data by one or more columns.
7. **Grouping and Aggregation:** Perform basic grouping and aggregation operations (`.groupby()`, `.agg()`).

Source Code: *(Provide Python code demonstrating all the Pandas functionalities mentioned in the procedure.)*

Input: A sample dataset (e.g., `sample_data.csv`).

Expected Output:

- Printed outputs of `.head()`, `.info()`, `.describe()`, `.isnull().sum()`, `.value_counts()`.
- Filtered and sorted DataFrames.
- Aggregated results from grouping operations.

Lab 9: Install and perform a simple Exploratory Data Analysis using Pandas

Title: Simple Exploratory Data Analysis (EDA) with Pandas

Aim: To perform a comprehensive exploratory data analysis (EDA) on a dataset using Pandas, identifying patterns, anomalies, and relationships, and preparing insights for further modeling.

Procedure:

1. **Load Dataset:** Load a dataset suitable for EDA.
2. **Data Cleaning:** Handle missing values and duplicates (as in Lab 2).
3. **Descriptive Statistics:** Generate detailed descriptive statistics.
4. **Univariate Analysis:** Analyze individual variables (distributions, outliers).
5. **Bivariate Analysis:** Explore relationships between two variables (e.g., scatter plots for numerical-numerical, box plots for numerical-categorical).
6. **Multivariate Analysis (Basic):** Consider relationships among more than two variables (e.g., pair plots, correlation matrices).
7. **Data Visualization:** Use Pandas' built-in plotting capabilities or integrate with Matplotlib/Seaborn for richer visualizations.
8. **Summarize Findings:** Document key insights and observations from the EDA.

Source Code: *(Provide Python code using Pandas for all EDA steps, potentially integrating Matplotlib/Seaborn for visualizations.)*

Input: A dataset that might require cleaning and offers opportunities for discovering patterns.

Expected Output:

- Cleaned DataFrame.
- Detailed statistical summaries.
- Various plots (histograms, box plots, scatter plots, correlation heatmap, etc.).
- A textual summary of key findings from the EDA.

Lab 10: Install, Import Scikit Learn and Explore Iris Dataset with Pandas for ML Modelling

Title: Introduction to Scikit-learn and Iris Dataset Exploration for ML Modelling

Aim: To install and import Scikit-learn, load and explore the famous Iris dataset using Pandas, and prepare it for basic machine learning modeling.

Procedure:

1. **Scikit-learn Installation:** Install the Scikit-learn library.
2. **Import Libraries:** Import `sklearn.datasets` and Pandas.
3. **Load Iris Dataset:** Load the Iris dataset using `load_iris()`.
4. **Convert to DataFrame:** Convert the Iris dataset (features and target) into a Pandas DataFrame for easier exploration.
5. **Initial Exploration:** Perform basic exploration on the Iris DataFrame (head, info, describe, value counts of target).
6. **Feature-Target Separation:** Separate features (X) and target (y) variables.
7. **Data Scaling (Optional/Conceptual):** Discuss the need for scaling and conceptually prepare for it (e.g., using `StandardScaler`).
8. **Train-Test Split:** Split the dataset into training and testing sets using `train_test_split`.

Source Code: *(Provide Python code using `sklearn.datasets`, Pandas, and `sklearn.model_selection.train_test_split`.)*

Input: No external input file needed, as Iris dataset is built-in.

Expected Output:

- Printed head, info, and descriptive statistics of the Iris DataFrame.
- Shape of X_train, X_test, y_train, y_test after splitting.

Lab 11: Install, Import Tensor flow and Keras . Create a Basic Neural Network with few layers .

Title: Building a Basic Neural Network with TensorFlow and Keras

Aim: To install TensorFlow and Keras, and construct a simple neural network with a few layers for a basic classification or regression task.

Procedure:

1. **TensorFlow/Keras Installation:** Install TensorFlow (which includes Keras).
2. **Import Libraries:** Import `tensorflow` and `keras`.
3. **Prepare Data:** Use a simple dataset (e.g., generated synthetic data, or a small preprocessed dataset like a subset of Iris or MNIST).
4. **Define Model Architecture:** Create a `Sequential` Keras model with:
    - An `Input` layer.
    - One or more `Dense` (fully connected) hidden layers with activation functions (e.g., 'relu').
    - An `Output` layer with an appropriate activation function (e.g., 'sigmoid' for binary classification, 'softmax' for multi-class classification, or linear for regression).
5. **Compile the Model:** Configure the model for training by specifying:
    - An `optimizer` (e.g., 'adam').
    - A `loss` function (e.g., 'binary_crossentropy', 'categorical_crossentropy', 'mse').
    - `metrics` to evaluate performance (e.g., 'accuracy').
6. **Train the Model:** Fit the model to the training data using `model.fit()`.
7. **Evaluate the Model:** Evaluate the model's performance on the test data using `model.evaluate()`.

Source Code: *(Provide Python code using TensorFlow and Keras to define, compile, train, and evaluate a basic neural network.)*

Input: A simple numerical dataset (e.g., NumPy arrays for features and target).

Expected Output:

- Model summary (`model.summary()`) showing layers and parameters.
- Training progress output during `model.fit()`.
- Evaluation results (loss and metrics) on the test set.

Lab 12: Install and perform a simple text processing using NLTK

Title: Simple Text Processing with NLTK

Aim: To install NLTK (Natural Language Toolkit) and perform fundamental text processing tasks such as tokenization, stemming, lemmatization, and stop word removal.

Procedure:

1. **NLTK Installation:** Install the NLTK library.
2. **Download NLTK Data:** Download necessary NLTK data (e.g., 'punkt', 'wordnet', 'stopwords').
3. **Import Modules:** Import relevant modules from `nltk`.
4. **Tokenization:** Tokenize a sample text into words and sentences.
5. **Stop Word Removal:** Remove common stop words from the tokenized text.
6. **Stemming:** Apply stemming (e.g., Porter Stemmer, Snowball Stemmer) to words.
7. **Lemmatization:** Apply lemmatization to words.
8. **Part-of-Speech Tagging (Optional):** Perform basic POS tagging.

Source Code: *(Provide Python code using NLTK for tokenization, stop word removal, stemming, and lemmatization.)*

Input: A sample string of text.

Expected Output:

- Tokenized words and sentences.
- Text after stop word removal.
- Stemmed words.
- Lemmatized words.

Lab 13: Install, Import OpenCV and Explore an Simple Image for Image Processing

Title: Basic Image Processing with OpenCV

Aim: To install and import OpenCV (Open Source Computer Vision Library) and perform fundamental image processing operations on a simple image.

Procedure:

1. **OpenCV Installation:** Install the OpenCV library (`opencv-python`).
2. **Import OpenCV:** Import `cv2`.
3. **Load Image:** Load a sample image (e.g., PNG, JPG) using `cv2.imread()`.
4. **Display Image:** Display the image using `cv2.imshow()`.
5. **Image Properties:** Get image properties (dimensions, number of channels).
6. **Color Space Conversion:** Convert the image to grayscale or other color spaces.
7. **Basic Manipulations:**
    - Resize the image.
    - Rotate the image.
    - Flip the image.
8. **Saving Image:** Save the modified image.

Source Code: *(Provide Python code using OpenCV for image loading, display, basic manipulation, and saving.)*

Input: A sample image file (e.g., `sample_image.jpg`) in the same directory as the script.

Expected Output:

- Original image displayed in a window.
- Modified images (grayscale, resized, rotated, flipped) displayed in separate windows.
- Confirmation of saved images.
- Printed image properties.

Lab 14: Install, Import Matplotlib . Explore all the Data Visualization Graphs .

Title: Exploring Data Visualization Graphs with Matplotlib

Aim: To install and import Matplotlib, and to explore and create various types of data visualization graphs to represent different aspects of data effectively.

Procedure:

1. **Matplotlib Installation:** Install the Matplotlib library.
2. **Import Matplotlib:** Import `matplotlib.pyplot`.
3. **Prepare Data:** Generate or use simple datasets suitable for different plot types.
4. **Create Various Plots:**
    - **Line Plot:** For visualizing trends over time or ordered data.
    - **Scatter Plot:** For showing relationships between two numerical variables.
    - **Bar Chart:** For comparing categorical data.
    - **Histogram:** For showing the distribution of a single numerical variable.
    - **Box Plot:** For visualizing distribution and identifying outliers.
    - **Pie Chart:** For showing proportions of a whole.
    - **Subplots:** Arrange multiple plots in a single figure.
5. **Customize Plots:** Add titles, labels, legends, and customize colors/styles.

Source Code: *(Provide Python code using Matplotlib to create and customize all the mentioned plot types.)*

Input: No external input, data will be generated or simple arrays.

Expected Output:

- Multiple distinct plot windows, each displaying a different type of data visualization graph with appropriate titles and labels.

Lab 15: Create all Data Visualization Plots using Matlotlib

Title: Comprehensive Data Visualization with Matplotlib

Aim: To apply Matplotlib to create a comprehensive set of data visualization plots on a given dataset, demonstrating proficiency in visual data storytelling.

Procedure:

1. **Load Dataset:** Use a slightly more complex dataset (e.g., one used in Lab 9 or 10) that allows for diverse visualizations.
2. **Data Preparation:** Ensure data is clean and ready for plotting.
3. **Strategic Plot Selection:** Choose appropriate plot types based on the data types and the insights to be conveyed (e.g., line plots for time series, scatter plots for correlations, heatmaps for matrices).
4. **Advanced Customization:** Apply advanced customization techniques:
    - Annotations and text.
    - Custom color maps.
    - Twin axes.
    - Different plot styles.
5. **Combine Plots:** Create complex figures with multiple subplots and inset plots.
6. **Export Plots:** Save plots in various formats (e.g., PNG, PDF).
7. **Interpret and Present:** Analyze the generated plots and summarize the insights they provide.

Source Code: *(Provide Python code using Matplotlib to create a variety of plots, focusing on advanced customization and combining multiple plots.)*

Input: A dataset (e.g., `data_for_viz.csv`) that has multiple numerical and categorical columns, potentially time series data.

Expected Output:

- A series of high-quality data visualization plots, potentially saved as image files.
- Each plot should be well-labeled, titled, and visually appealing, conveying specific insights from the data.
- A textual summary of the insights derived from the visualizations.