Practical 1: Data Wrangling I

Getting Started with Python

- **Introduction**
  - What is Python?: Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
  - It is mainly used for:
    - Software development
    - Mathematics (Data Science)
  - Python Syntax compared to other programming languages
    - Python was designed for readability, and has some similarities to the English language with influence from mathematics.
    - Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
    - Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.
- **Installation:**
  - Check for Installation
    - To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):
      - Windows+R
      - cmd
      - C:\Users\Your Name>python --version
  - If you find that you do not have Python installed on your computer, then you can download it for free from the following website: https://www.python.org/
  - **Installation**
    - Download Python:
      - Visit the official Python website: https://www.python.org/downloads/
      - Download the latest version of Python for your operating system (Windows, macOS, or Linux).
    - **Install Python:**
      - Run the installer and make sure to check the box that says "Add Python to PATH" before clicking on "Install Now". This will ensure that Python is accessible from the command line.
    - Verify Python Installation:
      - Windows+R

        - **Command Prompt** (CMD): cmd
        - C:\Users\Your Name>python --version
    - Install Jupyter Notebook with Libraries
        - Windows+R
        - **Command Prompt** (CMD): cmd
        - C:\Users\Your Name>pip install notebook
        - C:\Users\Your Name>pip install numpy pandas matplotlib seaborn scikit-learn scipy
    - Verifying Installation
        - pip list
    - Start Jupyter Notebook
        - After the installation is complete, run this command in the terminal to start Jupyter Notebook: jupyter notebook
  - **Installation Summary**
    - Install Python from python.org.
    - Install Jupyter Notebook using pip (pip install notebook)
    - Install all the libraries (pip install numpy pandas)
    - Start Jupyter Notebook with the command jupyter notebook
- **First Program**
  - print("Hello, World!")
- **Python Variables**
  - In Python, variables are created when you assign a value to it:

```
x = 5
y = "Hello, World!"
print("This is x:",x)
print(y)
```

- **Variable type**
  - You can get the data type of a variable with the type() function.
  - Variable names are case-sensitive.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

  - Assign multiple variables in one line.

```
x, y, z = "Orange", "Banana", "Cherry"
print(x,y,z)
```

- **Comments**
  - Comments start with a #, and Python will render the rest of the line as a comment:

```
#This is a comment.
print("Hello, World!")#This is also a comment.
```

- **Python Data Types**
  - String (str): *x = "Hello World"*
  - Integer (int): *x=5*
  - Float(float): *x=5.5*
  - List(list): *x = ["apple", "banana", "cherry"]*
  - Dictionary: *x = {"name" : "John", "age" : 36}*
    - Dictionary of Lists: Each column is a key, and its values are stored as lists

      ```
      data = {
        "Name": ["Alice", "Bob", "Charlie", "Patil"],
        "Age": [24, 30, 35, 25],
        "City": ["London", "Paris", "New York", "Mumbai"]
      }
      data
      ```

    - Convert to a **Pandas DataFrame** for advanced manipulation and analysis.

      ```
      import pandas as pd
      df = pd.DataFrame(data)
      print(df)
      type('Age') #data type of age
      df.dtypes #data type of data frame df
      ```

Python libraries:

1. pandas

pandas is a powerful library used for data manipulation and analysis. It provides data structures like **DataFrame** and **Series** to efficiently handle and manipulate large datasets.

- **DataFrame**: A two-dimensional table (similar to a spreadsheet or SQL table), with rows and columns.
- **Series**: A one-dimensional array-like object (like a column in a DataFrame).

Common Uses:

- Data cleaning, transformation, and aggregation.
- Handling missing data.
- Data selection and filtering.
- Merging, joining, and reshaping data.

Example:

```
import pandas as pd
```

```
# Create a DataFrame
data = {'Name': ['John', 'Alice', 'Bob'], 'Age': [28, 24, 22]}
df = pd.DataFrame(data)
# Display the DataFrame
print(df)
```

2. NumPy

NumPy (Numerical Python) is the foundational library for numerical computing in Python. It provides support for **large multi-dimensional arrays** and matrices, along with a collection of mathematical functions to operate on these arrays.

- **ndarray**: A multi-dimensional array object which is much faster than lists for numerical operations.
- Vectorization: Allows applying operations to entire arrays, which is more efficient than looping through elements.

Common Uses:

- Mathematical and logical operations on arrays.
- Random number generation.
- Linear algebra operations.

Example:

```
import numpy as np
# Create a NumPy array
arr = np.array([1, 2, 3, 4])
# Perform an operation (element-wise)
print(arr * 2)  # Output: [2 4 6 8]
```

3. Scikit-learn

Scikit-learn is a machine learning library that provides simple and efficient tools for data mining and data analysis. It is built on top of **NumPy**, **SciPy**, and **matplotlib** and is widely used for predictive data analysis.

- Provides algorithms for **classification**, **regression**, **clustering**, **dimensionality reduction**, and **model selection**.
- Includes utilities for data preprocessing, feature selection, and model evaluation.

Common Uses:

- Training machine learning models.

- Evaluating model performance (e.g., using confusion matrix, accuracy score).
- Preprocessing data (e.g., scaling, encoding).

Example:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
# Sample data
X = [[1], [2], [3], [4]]
y = [1, 2, 3, 4]
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
# Train a model
model = LinearRegression()
model.fit(X_train, y_train)
# Predict
print("X_train:", X_train, "X_test:", X_test)
print(y_train, y_test)
y_pred=model.predict(X_test)
print(y_test, y_pred)
```

4. matplotlib

matplotlib is a plotting library used for creating static, animated, and interactive visualizations in Python. It is highly customizable and can be used to create a wide range of plots and charts, including line plots, histograms, scatter plots, and bar charts.

- It works well with **NumPy** and **pandas** data structures.
- pyplot module is often used for simple plotting.

Common Uses:

- Creating various plots like line plots, scatter plots, bar charts, histograms, etc.
- Customizing plots (e.g., labels, titles, legends).

Example:

```
import matplotlib.pyplot as plt
# Create a simple plot
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]
plt.plot(x, y)
```

```
plt.title('Simple Plot')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.show()
```

5. seaborn

seaborn is built on top of **matplotlib** and provides a high-level interface for drawing attractive and informative statistical graphics. It is used for data visualization, especially in the context of statistical data analysis.

- It works seamlessly with **pandas** DataFrames and provides better default aesthetics than **matplotlib**.
- Supports complex plots such as **heatmaps**, **pair plots**, **violin plots**, and more.

Common Uses:

- Visualizing statistical relationships.
- Plotting distributions, categorical data, and correlation matrices.

Example:

```
import seaborn as sns
# Load an example dataset
tips = sns.load_dataset("tips")
# Create a scatter plot with seaborn
sns.scatterplot(data=tips, x="total_bill", y="tip")
plt.show()
```

6. nltk

nltk (Natural Language Toolkit) is a comprehensive library for **text processing and natural language processing (NLP)**. It includes tools for classification, tokenization, stemming, tagging, parsing, and more.

- It provides resources like **corpora**, **lexicons**, and pre-built functions to perform NLP tasks.
- Works well for building text analysis applications like **sentiment analysis**, **part-of-speech tagging**, and **named entity recognition**.

Common Uses:

- Tokenization (splitting text into words or sentences).
- Part-of-speech tagging.

- Stemming and Lemmatization.

Example:

```
import nltk
from nltk.tokenize import word_tokenize
# Sample text
text = "Hello, this is a test sentence."
# Tokenize the text
tokens = word_tokenize(text)
print(tokens)
```

7. scipy

scipy is a library that builds on **NumPy** and provides additional functionality for scientific and technical computing. It includes modules for optimization, integration, interpolation, eigenvalue problems, and more.

- It is especially useful for **mathematical**, **scientific**, and **engineering** tasks.
- It also includes statistical functions, optimization routines, and signal processing tools.

Common Uses:

- Solving optimization problems.
- Performing integration and differentiation.
- Signal and image processing.

Example:

```
from scipy import stats
# Calculate the mean and standard deviation of an array
data = [1, 2, 3, 4, 5]
mean = stats.tmean(data)
std_dev = stats.tstd(data)
print(f"Mean: {mean}, Standard Deviation: {std_dev}")
```

Summary:

- **pandas** and **NumPy** are primarily for data manipulation and numerical computing.
- **Scikit-learn** is focused on machine learning tasks.
- **matplotlib** and **seaborn** are used for data visualization.
- **nltk** is a toolkit for natural language processing (NLP).
- **scipy** provides advanced scientific and mathematical functions.

Each of these libraries is used in different phases of a data science pipeline, from **data preprocessing** and **analysis** to **visualization** and **modeling**.

Practical

Perform the following operations using Python on any open source dataset (e.g., data.csv)

1. Import all the required Python Libraries.
2. Locate an open source data from the web (e.g. https://www.kaggle.com). Provide a clear description of the data and its source (i.e., URL of the web site).
3. Load the Dataset into pandas data frame.
4. Data Preprocessing: check for missing values in the data using pandas isnull(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.
5. Data Formatting and Data Normalization: Summarize the types of variables by checking the data types (i.e., character, numeric, integer, factor, and logical) of the variables in the data set. If variables are not in the correct data type, apply proper type conversions.
6. Turn categorical variables into quantitative variables in Python.

In addition to the codes and outputs, explain every operation that you do in the above steps and explain everything that you do to import/read/scrape the data set.

Step 1: Import Required Libraries

```
import pandas as pd       # For data manipulation and analysis
import numpy as np        # For numerical computations
import matplotlib.pyplot as plt  # For visualisation (optional)
import seaborn as sb      # For enhanced visualisation (optional)
```

Step 2: Locate an Open Source Dataset

Find a dataset from an open-source platform Kaggle:

- Dataset: **Titanic Dataset**
- Description: This dataset contains information about Titanic passengers, such as age, sex, ticket class, and survival status.
- Source: Kaggle Titanic Dataset: https://www.kaggle.com/datasets/yasserh/titanic-dataset

Step 3: Load the Dataset

Download the dataset (Titanic-Dataset.csv), and load it into a Pandas DataFrame.

```
# Load the dataset
file_path = "C:/Users/Talha Ahmed/Desktop/My Practicals/Practical 1 Data Wrangling/Titanic-Dataset.csv"  # Update the path as per your file location
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
print(data.head())
```

Step 4: Data Preprocessing

Check for Missing Values

Use the isnull() function to identify missing values and describe() to get statistical summaries. Also check dimension of data.

```
# Check for missing values
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)

# Describe the data
print("\nData Statistics:\n", data.describe())

# Check dimensions
print("Data Dimensions:", data.shape)
```

Step 5: Data Formatting

Summarise Variable Types

Check data types and convert them if needed.

```
# Check data types
print("Data Types:\n", data.dtypes)

# Example: Convert 'Survived' to category
data['Survived'] = data['Survived'].astype('category')
data.dtypes       #check data type again

#Replace age null values to average age
data['Age'].isnull().sum()  #find number of null values
```

```
mean_age = data['Age'].mean()  #find mean
mean_age #show mean
# Replace missing values in 'Age' with the mean
data['Age'] = data['Age'].fillna(mean_age)
# drop cabin as it has 687 missing values
data=data.drop(columns=['Cabin'])
```

Step 6: Convert Categorical Variables

Convert categorical variables (e.g., Sex,) into quantitative ones using one-hot encoding.

```
# One-hot encode categorical variables
data = pd.get_dummies(data, columns=['Sex'], drop_first=True)

# Display the updated DataFrame
print(data.head())
# ----------OR----------
data["Sex"].replace({"female":0,"male":1})   #just replace values
data
```

Bonus: Export Cleaned Data

```
# Specify the output file path
output_file_path = "C:/Users/Talha Ahmed/Desktop/My Practicals/Practical 1 Data Wrangling/clean_titanic_data.csv"

# Export the grouped data to a CSV file
data.to_csv(output_file_path)
```

Bonus: Some Graphs

1. **BoxPlot**

```
import seaborn as sns
import matplotlib.pyplot as plt

# Box plot for 'Age'
sns.boxplot(x=data['Age'], color="skyblue")
plt.title("Box Plot of Age")
plt.xlabel("Age")
plt.show()
```

2. **Violin Plot**

```
sns.violinplot(x=data['Age'], color="skyblue")
plt.title("Violin Plot of Age")
plt.xlabel("Age")
plt.show()
```

3. **Histogram**

```
import matplotlib.pyplot as plt
# Histogram for Age
plt.hist(data['Age'], bins=15, color='skyblue', edgecolor='black')
plt.axvline(data['Age'].mean(), color='red', linestyle='dashed', linewidth=1,
label='Mean')
plt.axvline(data['Age'].mean() - data['Age'].std(), color='orange',
linestyle='dashed', linewidth=1, label='-1 SD')
plt.axvline(data['Age'].mean() + data['Age'].std(), color='orange',
linestyle='dashed', linewidth=1, label='+1 SD')
plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.legend()
plt.show()
```