LAB Assignment 1:

**Data Wrangling, I**

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

1. Import all the required Python Libraries.

   - import **pandas** as **pd**

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.

   - **df = pd.read_csv('hepatitis.csv')**
   - **df.head(10)**

4. Data Preprocessing: check for missing values in the data using pandas insult(), describe() function to get some initial statistics. Provide variable descriptions. Types of variables etc. Check the dimensions of the data frame.

   - **df.isna().sum()**
   - **df.isna().sum()/len(df)*****100 (**finding % of missing Value**)**
   - **df.dropna(axis=1)**
   - **df.describe()**
   - **df.ndim()**
   - **df.dtypes**

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.

   - There are multiple normalization techniques in statistics. In this article, we will cover the most important ones:
     - The maximum absolute scaling
     - The min-max feature scaling
     - The z-score method
     - The robust scaling

1. **The maximum absolute scaling**
   - The maximum absolute scaling rescales each feature between -1 and 1 by dividing every observation by its maximum absolute value.

$$x_{scaled} = \frac{x}{max(|x|)}$$

   - We can apply the **maximum absolute scaling** in **Pandas** using the **.max()** and **.abs()** methods, as shown below.
   - Example:

```
df_max_scaled = df.copy()

# apply normalization techniques
for column in df_max_scaled.columns:
   df_max_scaled[column] = df_max_scaled[column]/ df_max_scaled[column].abs().max()

# view normalized data
display(df_max_scaled)
```

2. **The min-max feature scaling**

- The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range.
- We can apply the min-max scaling in Pandas using the .min() and .max() methods.

```
# copy the data
df_min_max_scaled = df.copy()

# apply normalization techniques
for column in df_min_max_scaled.columns:
   df_min_max_scaled[column] = (df_min_max_scaled[column] - df_min_max_scaled[column].min()) /
   (df_min_max_scaled[column].max() - df_min_max_scaled[column].min())

# view normalized data
print(df_min_max_scaled)
```

3. **Using The z-score method**

- The z-score method (often called standardization) transforms the info into distribution with a mean of 0 and a typical deviation of 1.
- Each standardized value is computed by subtracting the mean of the corresponding feature then dividing by the quality deviation.

```
# copy the data
df_z_scaled = df.copy()

# apply normalization techniques
for column in df_z_scaled.columns:
         df_z_scaled[column] = (df_z_scaled[column] -  df_z_scaled[column].mean()) /
         df_z_scaled[column].std()

# view normalized data
display(df_z_scaled)
```

4. **The Robust Scaling**

- In robust scaling, we scale each feature of the data set by subtracting the median and then dividing by the interquartile range.
- The interquartile range (IQR) is defined as the difference between the third and the first quartile and represents the central 50% of the data.

- Mathematically the robust scaler can be expressed as:

$$x_{rs} = \frac{x_i - Q_2(x)}{Q_3(x) - Q_1(x)}$$

- where **Q1(x)** is the **first quartile** of the attribute x, **Q2(x)** is the **median**, and **Q3(x)** is the **third quartile**.

```
from
sklearn.preprocessing
import RobustScaler
```

```
# create a scaler object
scaler = RobustScaler()
# fit and transform the data
df_robust = pd.DataFrame(scaler.fit_transform(df_cars),
columns=df_cars.columns)



df_robust
```

6. Turn categorical variables into quantitative variables in Python.

1. Gender : replace 0 to males, 1 to females
2. Getting output as:

```
      name  episodes  gender  female  male
0  Sheldon        42    male       0     1
1    Penny        24  female       1     0
```

- import pandas as pd
- data = {'name': ['Sheldon', 'Penny', 'Amy', 'Penny', 'Raj', 'Sheldon'],
- 'episodes': [42, 24, 31, 29, 37, 40],
- 'gender': ['male', 'female', 'female', 'female', 'male', 'male']}
- df = pd.DataFrame(data)
- df_gender = pd.get_dummies(df['gender'])
- df_new = pd.concat([df, df_gender], axis=1)
- print(df_new)

7. 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.

Assignment 2: **Data Wrangling II**

**Create an "Academic performance" dataset of students and perform the following operations using Python.**

Creating dataset with missing values:

- import pandas as pd
- import numpy as np
- df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f', 'h'],columns=['one', 'two', 'three'])
- df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
- print df

1. Scan all variables for missing values and inconsistencies. If there are missing values and/or inconsistencies, use any of the suitable techniques to deal with them.
   a. Finding missing values
      - print df['one'].isnull()

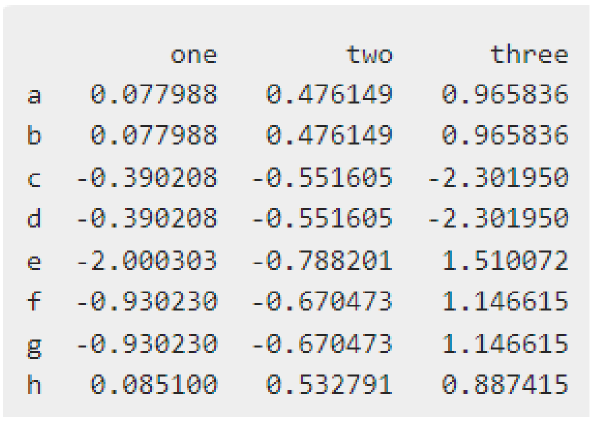
   b. Replace NaN with a Scalar Value
      - import pandas as pd
      - import numpy as np
      - df = pd.DataFrame(np.random.randn(3, 3), index=['a', 'c', 'e'],columns=['one',
      - 'two', 'three'])
      - df = df.reindex(['a', 'b', 'c'])
      - print df
      - print ("NaN replaced with '0':")
      - print df.fillna(0)

   c. Fill NA Forward and Backward
      - Using the concepts of filling discussed in the ReIndexing Chapter we will fill the missing values.

| Sr.No | • Method & Action |
|---|---|
| 1 | • pad/fill<br>• Fill methods Forward |
| 2 | • bfill/backfill<br>• Fill methods Backward |

**Example 1**

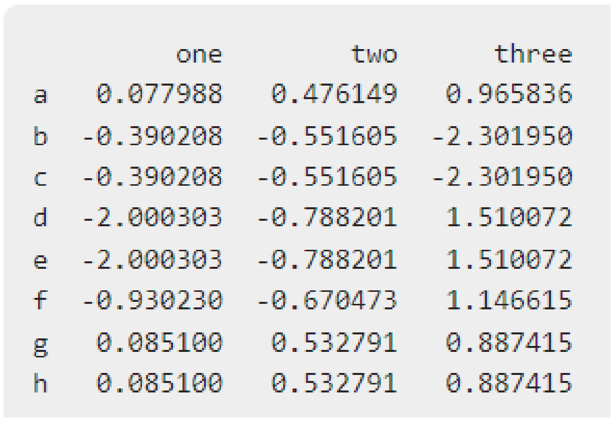
- import pandas as pd
- import numpy as np
- df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],columns=['one', 'two', 'three'])
- df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
- print df.fillna(method='pad')



```
       one       two     three
a   0.077988  0.476149  0.965836
b   0.077988  0.476149  0.965836
c  -0.390208 -0.551605 -2.301950
d  -0.390208 -0.551605 -2.301950
e  -2.000303 -0.788201  1.510072
f  -0.930230 -0.670473  1.146615
g  -0.930230 -0.670473  1.146615
h   0.085100  0.532791  0.887415
```

**Example 2**

- import pandas as pd
- import numpy as np
- df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],columns=['one', 'two', 'three'])
- df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
- print df.fillna(method='backfill')



```
       one       two     three
a   0.077988  0.476149  0.965836
b  -0.390208 -0.551605 -2.301950
c  -0.390208 -0.551605 -2.301950
d  -2.000303 -0.788201  1.510072
e  -2.000303 -0.788201  1.510072
f  -0.930230 -0.670473  1.146615
g   0.085100  0.532791  0.887415
h   0.085100  0.532791  0.887415
```

Example 3: Dropping Missing Values

- import pandas as pd
- import numpy as np
- df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f','h'],columns=['one', 'two', 'three'])
- df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
- print df.dropna()

2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.

**Quantile:**

- pandas **dataframe.quantile()** function return values at the given quantile over requested axis, a numpy.percentile.

- **Syntax:** DataFrame.quantile(q=0.5, axis=0, numeric_only=True, interpolation='linear')
  **Parameters :**
  - float or array-like, default 0.5 (50% quantile). 0 <= q <= 1, the quantile(s) to compute
  - **axis :** [{0, 1, 'index', 'columns'} (default 0)] 0 or 'index' for row-wise, 1 or 'columns' for column-wise
  - **numeric_only :** If False, the quantile of datetime and timedelta data will be computed as well
  - **interpolatoin :** {'linear', 'lower', 'higher', 'midpoint', 'nearest'}

**Example 1: Using Quantiles**

- **calculate and print the interquartile range for each of the variables in the dataset.**
  - Q1 = df.quantile(0.25)
  - Q3 = df.quantile(0.75)
  - IQR = Q3 - Q1
  - print(IQR)
- **The code below generates an output with the 'True' and 'False' values**
  - print(df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))

**Example 2: Identifying Outliers with Skewness**

- print(df['Income'].skew())
- df['Income'].describe()

**Example 3: Identifying Outliers with Visualization**

- plt.boxplot(df["Loan_amount"])
- plt.show()
- df.boxplot(column='Loan_amount', by='approval_status')

**Example 4: Histogram**

**Example 5: Scatter**

**Outlier Treatment**

### Solution 1: Quantile-based Flooring and Capping

- In this technique, we will do the flooring (e.g., the 10th percentile) for the lower values and capping (e.g., the 90th percentile) for the higher values
  - print(df['Income'].quantile(0.10))
  - print(df['Income'].quantile(0.90))
- Now we will remove the outliers, as shown in the lines of code below
  - df["Income"] = np.where(df["Income"] <2960.0, 2960.0,df['Income'])
  - df["Income"] = np.where(df["Income"] >12681.0, 12681.0,df['Income'])
  - print(df['Income'].skew())
-

3. Apply data transformations on at least one of the variables. The purpose of this transformation should be one of the following reasons: to change the scale for better understanding of the variable, to convert a non-linear relation into a linear one, or to

decrease the skewness and convert the distribution into a normal distribution. Reason and document your approach properly.