Technological Institute of the Philippines
Computer Engineering Department
Quezon city Campus

Hands-on Activity 10.1 Data Analysis using Python

Course: CPE 311	**Program: BSCpE**
Course Title: Computational Thinking with Python	**Date Performed:** April 4 , 2024
Section: BSCPE22S3	**Date Submitted:** April 4 , 2024
Student Name: Juliann Vincent B. Quibral	**Instructor's Name:** Engr. Roman Richard
Partner Name: Jhillian M. Cabos	

Import Modules

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

Load the dataset into pandas dataframe

```
df = pd.read_csv('cStick.csv')
print("Original Dataframe:\n", df)
```

```
Original Dataframe:
       Distance  Pressure      HRV  Sugar level    SpO2  Accelerometer  \
0       25.540       1.0  101.396       61.080  87.770            1.0
1        2.595       2.0  110.190       20.207  65.190            1.0
2       68.067       0.0   87.412       79.345  99.345            0.0
3       13.090       1.0   92.266       36.180  81.545            1.0
4       69.430       0.0   89.480       80.000  99.990            0.0
...        ...       ...      ...          ...     ...            ...
2034     5.655       2.0  116.310      162.242  71.310            1.0
2035     9.660       2.0  124.320      177.995  79.320            1.0
2036    15.220       1.0   93.828       40.440  82.610            1.0
2037     9.120       2.0  123.240      175.871  78.240            1.0
2038    62.441       0.0   78.876       76.435  96.435            0.0

      Decision
0            1
1            2
2            0
3            1
4            0
...        ...
2034         2
2035         2
2036         1
2037         2
2038         0

[2039 rows x 7 columns]
```

```
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2039 entries, 0 to 2038
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Distance       2039 non-null   float64
 1   Pressure       2039 non-null   float64
 2   HRV            2039 non-null   float64
 3   Sugar level    2039 non-null   float64
 4   SpO2           2039 non-null   float64
 5   Accelerometer  2039 non-null   float64
 6   Decision       2039 non-null   int64
dtypes: float64(6), int64(1)
memory usage: 111.6 KB
None
          Distance     Pressure          HRV  Sugar level         SpO2  \
count  2039.000000  2039.000000  2039.000000  2039.000000  2039.000000
```

```
mean     28.694527     0.988720    95.657002    72.909243    83.563649
std      23.773644     0.815918    17.576499    46.940110    11.111592
min       0.000000     0.000000    60.000000    10.000000    60.000000
25%       7.642500     0.000000    82.418000    40.230000    75.285000
50%      20.560000     1.000000    97.238000    69.960000    85.280000
75%      55.205500     2.000000   109.695000    77.612500    92.692500
max      69.981000     2.000000   124.980000   179.293000    99.990000

       Accelerometer    Decision
count    2039.000000  2039.000000
mean        0.661599     0.988720
std         0.473282     0.815918
min         0.000000     0.000000
25%         0.000000     0.000000
50%         1.000000     1.000000
75%         1.000000     2.000000
max         1.000000     2.000000
```

Data Cleaning

(**Data Cleaning**) *Check for missing Values*

This step involves examining the dataframe to identify any missing values in each column. It helps in understanding the completeness of the dataset.

```
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)
```

```
Missing Values:
 Distance        0
Pressure        0
HRV             0
Sugar level     0
SpO2            0
Accelerometer   0
Decision        0
dtype: int64
```

```
1
```

(**Data Cleaning**) *Check for missing Values*

If missing values are found, they can be filled using various techniques. In this example, missing values are filled with the mean value of each column, ensuring data completeness

```
df.fillna(df.mean(), inplace=True)
print("Dataframe after filling missing values with mean:\n", df)
```

```
Dataframe after filling missing values with mean:
       Distance  Pressure      HRV  Sugar level    SpO2  Accelerometer  \
0       25.540       1.0  101.396       61.080  87.770            1.0
1        2.595       2.0  110.190       20.207  65.190            1.0
2       68.067       0.0   87.412       79.345  99.345            0.0
3       13.090       1.0   92.266       36.180  81.545            1.0
4       69.430       0.0   89.480       80.000  99.990            0.0
...        ...       ...      ...          ...     ...            ...
2034     5.655       2.0  116.310      162.242  71.310            1.0
2035     9.660       2.0  124.320      177.995  79.320            1.0
2036    15.220       1.0   93.828       40.440  82.610            1.0
2037     9.120       2.0  123.240      175.871  78.240            1.0
2038    62.441       0.0   78.876       76.435  96.435            0.0

      Decision
0            1
1            2
2            0
3            1
4            0
...        ...
2034         2
2035         2
2036         1
2037         2
```

```
2038          0

[2039 rows x 7 columns]
```

(**Data Cleaning**) *Check for Duplicate Rows*

Duplicate rows may skew analysis results, so it's important to identify and handle them. This step checks for duplicate rows in the dataframe.

```
duplicate_rows = df[df.duplicated()]
print("Duplicate Rows:\n", duplicate_rows)
```

```
Duplicate Rows:
 Empty DataFrame
Columns: [Distance, Pressure, HRV, Sugar level, SpO2, Accelerometer, Decision ]
Index: []
```

(**Data Cleaning**) *Drop Duplicate Rows*

Duplicate rows, if found, are removed from the dataframe. This ensures each observation is unique and prevents duplication biases in analysis.

```
df.drop_duplicates(inplace=True)
print("Dataframe after dropping duplicate rows:\n", df)
```

```
Dataframe after dropping duplicate rows:
       Distance  Pressure      HRV  Sugar level    SpO2  Accelerometer  \
0       25.540       1.0  101.396       61.080  87.770            1.0
1        2.595       2.0  110.190       20.207  65.190            1.0
2       68.067       0.0   87.412       79.345  99.345            0.0
3       13.090       1.0   92.266       36.180  81.545            1.0
4       69.430       0.0   89.480       80.000  99.990            0.0
...        ...       ...      ...          ...     ...            ...
2034     5.655       2.0  116.310      162.242  71.310            1.0
2035     9.660       2.0  124.320      177.995  79.320            1.0
2036    15.220       1.0   93.828       40.440  82.610            1.0
2037     9.120       2.0  123.240      175.871  78.240            1.0
2038    62.441       0.0   78.876       76.435  96.435            0.0

      Decision
0            1
1            2
2            0
3            1
4            0
...        ...
2034         2
2035         2
2036         1
2037         2
2038         0

[2039 rows x 7 columns]
```

(**Data Cleaning**) *Check for Outliers*

Outliers are data points that significantly differ from other observations in the dataset. This step identifies outliers using z-scores, which measure how many standard deviations a data point is from the mean.

```
from scipy.stats import zscore
z_scores = zscore(df.select_dtypes(include=np.number))
abs_z_scores = np.abs(z_scores)
outliers = (abs_z_scores > 3).all(axis=1)
print("Outliers:\n", df[outliers])
```

```
Outliers:
 Empty DataFrame
Columns: [Distance, Pressure, HRV, Sugar level, SpO2, Accelerometer, Decision ]
Index: []
```

(**Data Cleaning**) *View Data Types*

It's crucial to ensure that the data types of each column are appropriate for analysis. This step checks the data types of columns in the dataframe.

```
print("Data Types:\n", df.dtypes)
```

```
Data Types:
 Distance        float64
Pressure        float64
HRV             float64
Sugar level     float64
SpO2            float64
Accelerometer   float64
Decision          int64
dtype: object
```

(**After Data Cleaning**) *Check for Data Values*

Descriptive statistics provide summaries of the dataset's main characteristics. This step uses the describe() function to compute basic statistics like mean, median, and quartiles for numerical columns.

```
print("\nDf information after preprocessing:\n", df.info())
print("\nDf statistics after preprocessing:\n", df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2039 entries, 0 to 2038
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Distance       2039 non-null   float64
 1   Pressure       2039 non-null   float64
 2   HRV            2039 non-null   float64
 3   Sugar level    2039 non-null   float64
 4   SpO2           2039 non-null   float64
 5   Accelerometer  2039 non-null   float64
 6   Decision       2039 non-null   int64
dtypes: float64(6), int64(1)
memory usage: 127.4 KB

Df information after preprocessing:
 None

Df statistics after preprocessing:
          Distance     Pressure          HRV  Sugar level         SpO2  \
count  2039.000000  2039.000000  2039.000000  2039.000000  2039.000000
mean     28.694527     0.988720    95.657002    72.909243    83.563649
std      23.773644     0.815918    17.576499    46.940110    11.111592
min       0.000000     0.000000    60.000000    10.000000    60.000000
25%       7.642500     0.000000    82.418000    40.230000    75.285000
50%      20.560000     1.000000    97.238000    69.960000    85.280000
75%      55.205500     2.000000   109.695000    77.612500    92.692500
max      69.981000     2.000000   124.980000   179.293000    99.990000

       Accelerometer    Decision
count    2039.000000  2039.000000
mean        0.661599     0.988720
std         0.473282     0.815918
min         0.000000     0.000000
25%         0.000000     0.000000
50%         1.000000     1.000000
75%         1.000000     2.000000
max         1.000000     2.000000
```

```
print("Descriptive statistics of the dataframe:\n", df.describe())
```

```
Descriptive statistics of the dataframe:
          Distance     Pressure          HRV  Sugar level         SpO2  \
count  2039.000000  2039.000000  2039.000000  2039.000000  2039.000000
mean     28.694527     0.988720    95.657002    72.909243    83.563649
std      23.773644     0.815918    17.576499    46.940110    11.111592
min       0.000000     0.000000    60.000000    10.000000    60.000000
25%       7.642500     0.000000    82.418000    40.230000    75.285000
50%      20.560000     1.000000    97.238000    69.960000    85.280000
75%      55.205500     2.000000   109.695000    77.612500    92.692500
max      69.981000     2.000000   124.980000   179.293000    99.990000

       Accelerometer    Decision
count    2039.000000  2039.000000
```

mean	0.661599	0.988720
std	0.473282	0.815918
min	0.000000	0.000000
25%	0.000000	0.000000
50%	1.000000	1.000000
75%	1.000000	2.000000
max	1.000000	2.000000

Perform Correlation Analysis

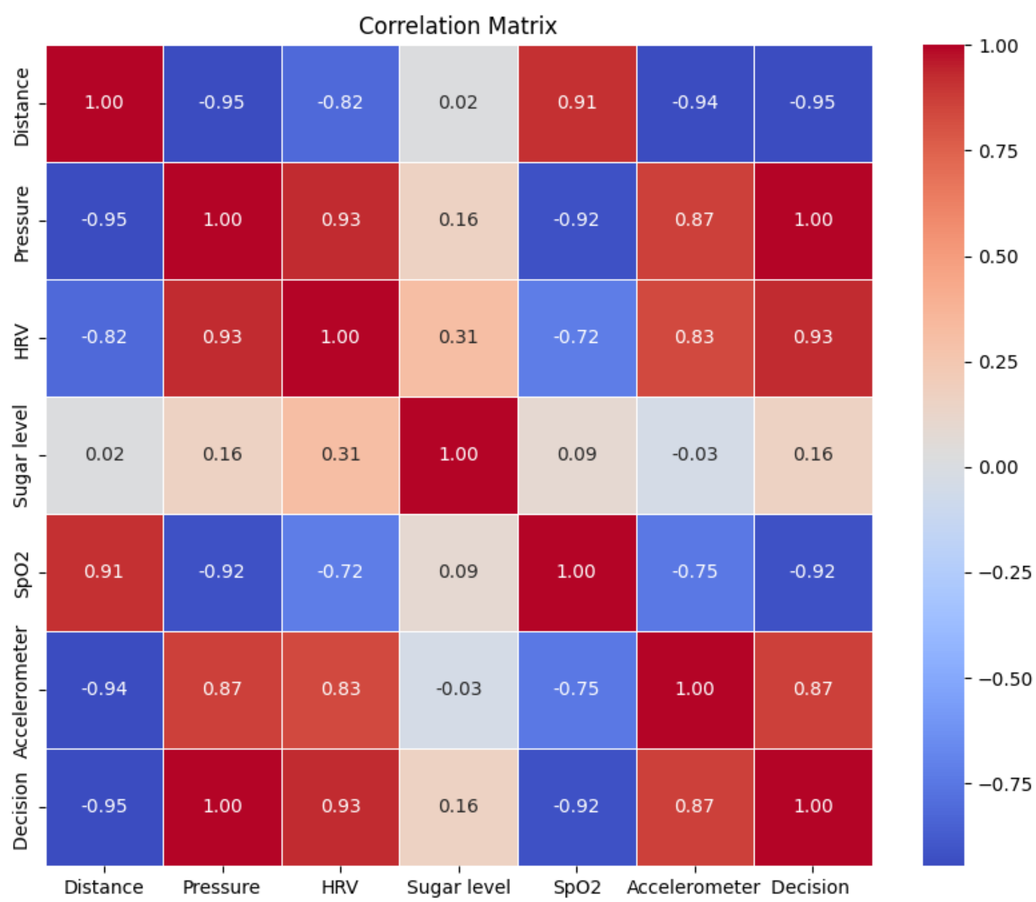
Correlation analysis measures the strength and direction of the linear relationship between two variables. It helps in understanding how variables are related to each other in the dataset.

```
correlation_matrix = df.corr()
```

Visualize the correlation matrix using heatmap

Visualizing the correlation matrix using a heatmap provides a clear and concise representation of the relationships between variables. It helps in identifying patterns and dependencies within the data.

```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



Conclusion

We handled missing values and duplicates to guarantee data integrity after carefully reviewing the dataset. We cleaned up the dataset for analysis by removing duplicates and used column means to fill in any missing values. Next, in order to make sure our study represents significant patterns, we used z-scores to identify outliers. The dependability of our dataset was further reinforced by checking the data kinds. While correlation analysis highlighted links between variables, descriptive statistics offered insight into numerical properties. Using a heatmap to visualize relationships improved our comprehension of data trends. This method, which is similar to priming a canvas before painting, established the groundwork for perceptive investigation and interpretation, which are essential for both academic and professional pursuits.

Furthermore, regarding the dataset from which we extracted the correlations, improving fall prediction and detection methods for senior citizens depends critically on the identification of correlations within the dataset. It makes it possible to optimize the models and algorithms that are employed in products like cStick, increasing their efficacy and accuracy. Understanding correlations also makes it easier to customize solutions to meet each person's particular needs, which eventually advances the area of elder care technology.

Link for dataset: https://www.kaggle.com/datasets/laavanya/elderly-fall-prediction-and-detection