Department of
Computer Science and Engineering

Title: Data Preprocessing Techniques

Data Mining LAB

CSE 424

Green University of Bangladesh

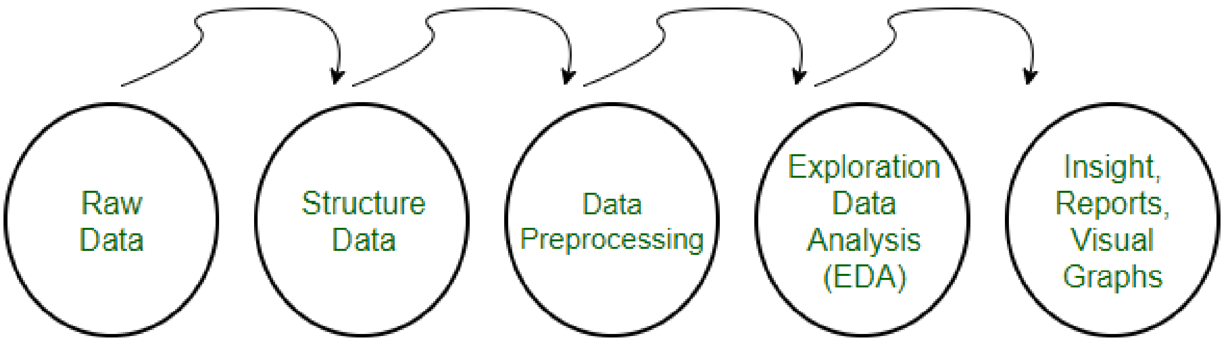
1 Objective(s)

- To transform raw data more easily and effectively processed in data mining, machine learning and other data science tasks.

2 Data Preprocessing

Data preprocessing is the process of transforming raw data into an understandable format.Before applying machine learning or data mining algorithms, the quality of the data should be checked by the following-

- **Accuracy:** To check whether the data entered is correct or not.
- **Completeness:** To check whether the data is available or not recorded.
- **Consistency:** To check whether the same data is kept in all the places that do or do not match.
- **Timeliness:** The data should be updated correctly.
- **Believability:** The data should be trustable.
- **Interpretability:** The understandability of the data.


Raw Data
Structure Data
Data Preprocessing
Exploration Data Analysis (EDA)
Insight, Reports, Visual Graphs

Data Preprocessing Steps

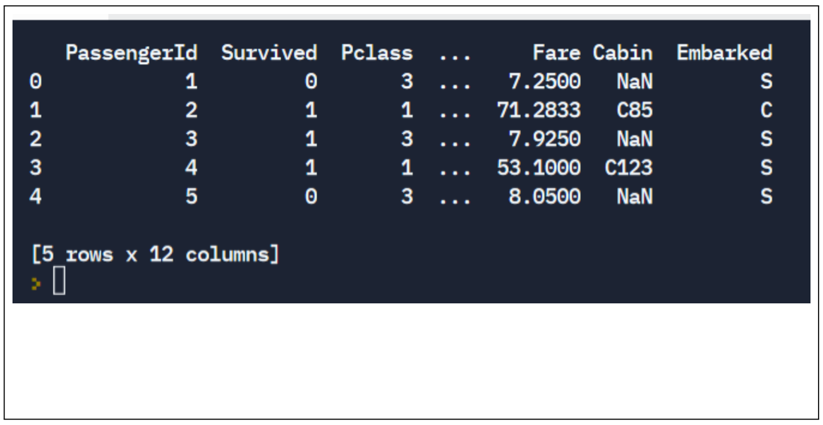
Before diving into the preprocessing, we need a dataset to be loaded. We will be working with the Titanic Dataset from Kaggle. For downloading the dataset, use the following link- https://www.kaggle.com/c/titanic

Implementation in Python

```python
import pandas as pd
import numpy as np
df = pd.read_csv("titanic_dataset.csv")

print(df.head())
```

Input/Output

Output of the programs is given below.



```
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S

[5 rows x 12 columns]
```

See that the dataset contains many columns like PassengerId, Name, Age etc. We are going to be deleting the unnecessary columns such as Name, Ticket, PassengerId, Cabin and Embarked.

Implementation in Python

```
df.drop("Name",axis=1,inplace=True)
df.drop("Ticket",axis=1,inplace=True)
df.drop("PassengerId",axis=1,inplace=True)
df.drop("Cabin",axis=1,inplace=True)
df.drop("Embarked",axis=1,inplace=True)
```

2.1 Data cleaning

Data cleaning is the process to remove incorrect data, incomplete data and inaccurate data from the datasets, and it also replaces the missing values. There are some techniques in data cleaning.

2.1.1 Handling missing values

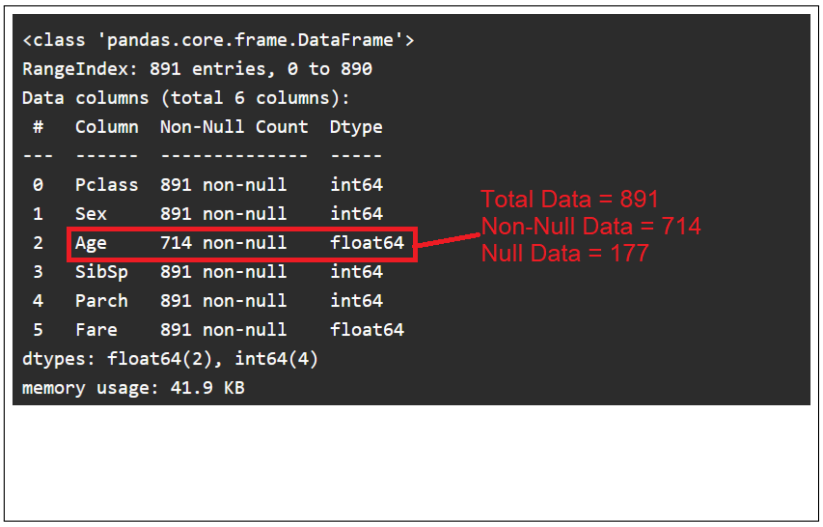
Missing values are usually represented in the form of Nan or null or None in the dataset. ".info()" the function can be used to give information about the dataset. This will provide you with the column names along with the number of non – null values in each column.

Implementation in Python

```
df.info()
```

Input/Output

Output of the programs is given below.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Pclass  891 non-null    int64
 1   Sex     891 non-null    int64
 2   Age     714 non-null    float64
 3   SibSp   891 non-null    int64
 4   Parch   891 non-null    int64
 5   Fare    891 non-null    float64
dtypes: float64(2), int64(4)
memory usage: 41.9 KB
```

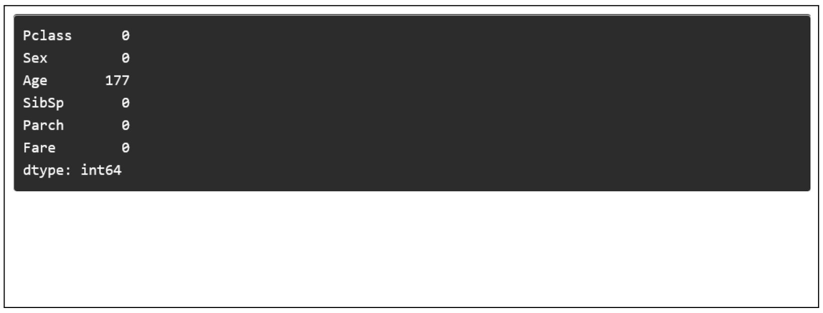
See that there are null values in the column Age. The second way of finding whether we have null values in the data is by using the "isnull()" function.

Implementation in Python

```
print(df.isnull().sum())
```

Input/Output

Output of the programs is given below.



```
Pclass      0
Sex         0
Age       177
SibSp       0
Parch       0
Fare        0
dtype: int64
```

See that all the null values in the dataset are in the column "Age". Let's try fitting the data using following techniques.

2.1.2 Filling the Missing Values – Imputation:

In this case, we will be filling the missing values with a certain number. The possible ways to do this are:

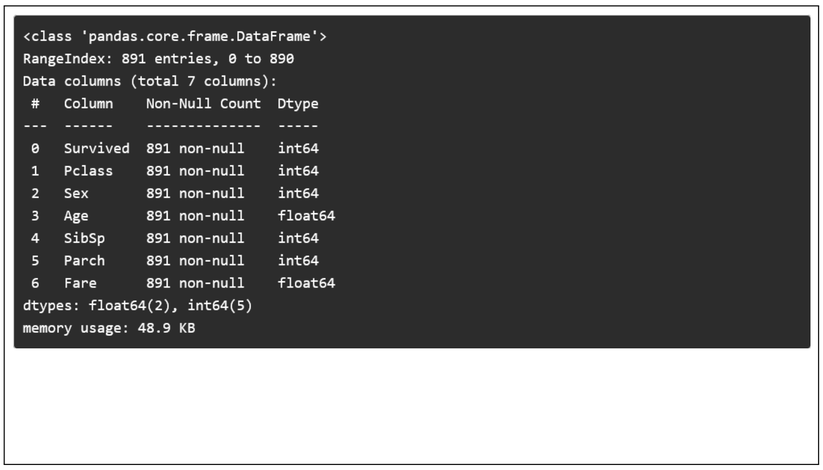
1. Filling the missing data with the mean or median value if it's a numerical variable.
2. Filling the missing data with mode if it's a categorical value.
3. Filling the numerical value with 0 or -999, or some other number that will not occur in the data. This can be done so that the machine can recognize that the data is not real or is different.
4. Filling the categorical value with a new type for the missing values.

You can use the "fillna()" function to fill the null values in the dataset.

Implementation in Python

```
updated_df = df
updated_df['Age']=updated_df['Age'].fillna(updated_df['Age'].mean())
updated_df.info()
```

Input/Output



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 48.9 KB
```

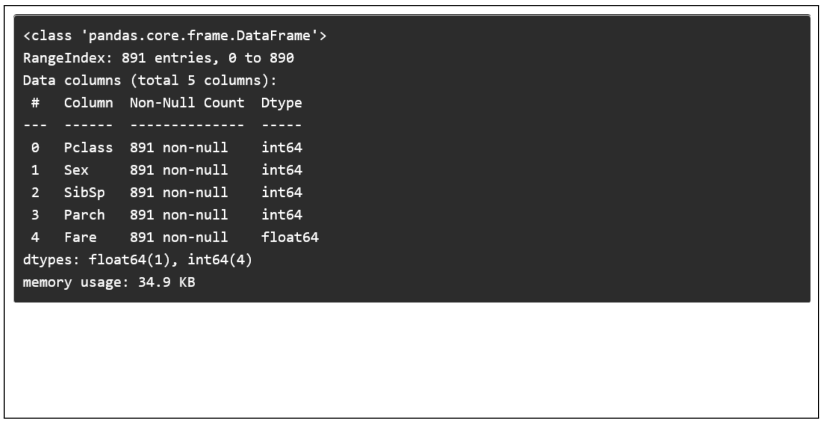
2.1.3 Deleting the columns with missing data

In this case, let's delete the column, Age and then fit the model and check for accuracy. But this is an extreme case and should only be used when there are many null values in the column.

Implementation in Python

```
updated_df = df.dropna(axis=1)
updated_df.info()
```

Output of the programs is given below.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Pclass  891 non-null    int64
 1   Sex     891 non-null    int64
 2   SibSp   891 non-null    int64
 3   Parch   891 non-null    int64
 4   Fare    891 non-null    float64
dtypes: float64(1), int64(4)
memory usage: 34.9 KB
```

The problem with this method is that we may lose valuable information on that feature, as we have deleted it completely due to some null values.

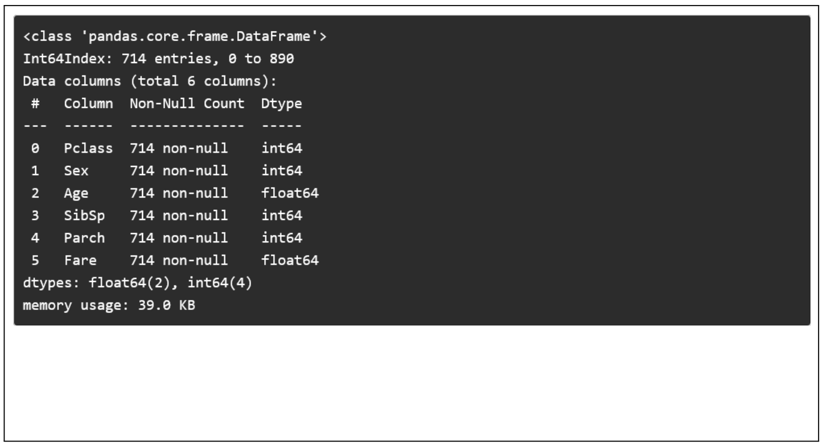
2.1.4 Deleting the row with missing data:

If there is a certain row with missing data, then you can delete the entire row with all the features in that row.
axis=1 is used to drop the column with 'NaN' values.
axis=0 is used to drop the row with 'NaN' values.

Implementation in Python

```
updated_df = newdf.dropna(axis=0)

y1 = updated_df['Survived']
updated_df.drop("Survived",axis=1,inplace=True)

updated_df.info()
```

Input/Output



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 714 entries, 0 to 890
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Pclass  714 non-null    int64  
 1   Sex     714 non-null    int64  
 2   Age     714 non-null    float64
 3   SibSp   714 non-null    int64  
 4   Parch   714 non-null    int64  
 5   Fare    714 non-null    float64
dtypes: float64(2), int64(4)
memory usage: 39.0 KB
```

2.1.5 Handling noisy data:

Noisy generally means random error or containing unnecessary data points. Here are some of the methods to handle noisy data.

- **Binning method:** In this method, the data is first sorted and then the sorted values are distributed into a number of buckets or bins. As binning methods consult the neighbourhood of values, they perform local smoothing. There are three approaches to performing Binning method –
  1. Sort the array of a given data set.
  2. Divides the range into N intervals, each containing the approximately same number of samples(Equal-depth partitioning).
  3. Store mean/ median/ boundaries in each row.

Python implementation: Try Yourself!!!

2.2 Data transformation

The change made in the format or the structure of the data is called data transformation. This step can be simple or complex based on the requirements. There are some methods in data transformation.

2.2.1 Discretization

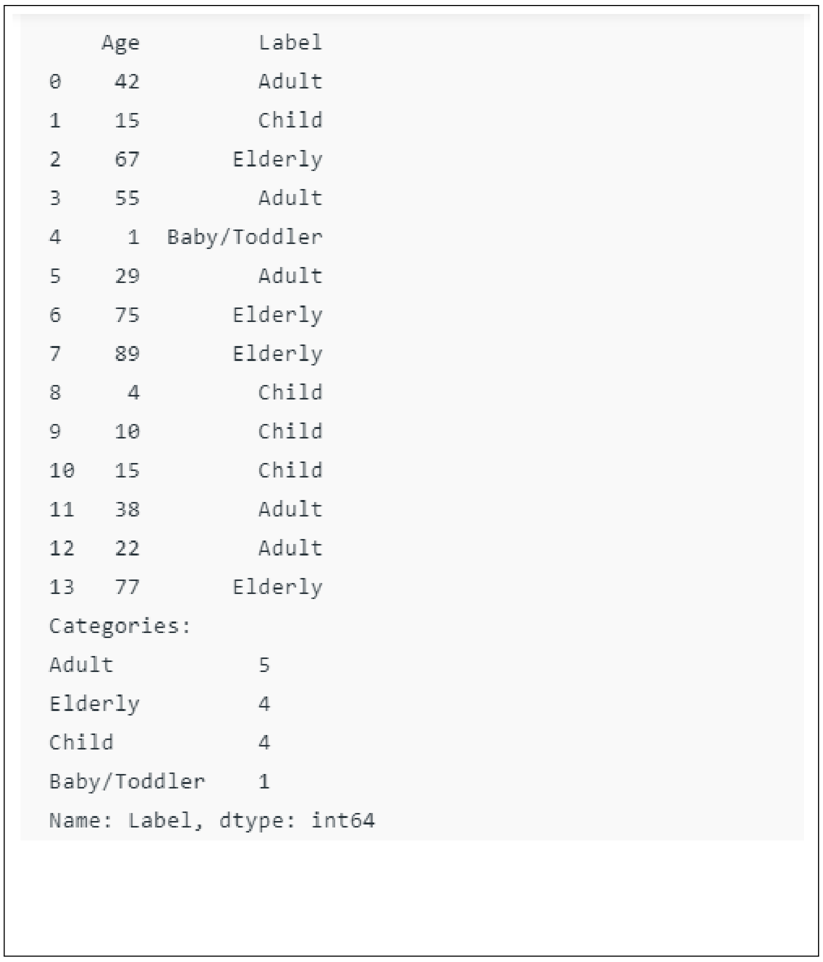
The continuous data here is split into intervals. Discretization reduces the data size. For example, rather than specifying the class time, we can set an interval like (3 pm-5 pm, 6 pm-8 pm).

Implementation in Python

```

# Importing pandas and numpy libraries
import pandas as pd
import numpy as np

# Creating a dummy DataFrame of 15 numbers randomly
# ranging from 1-100 for age
df = pd.DataFrame({'Age': [42, 15, 67, 55, 1, 29, 75, 89, 4,
                           10, 15, 38, 22, 77]})

# Printing DataFrame Before sorting Continuous
# to Categories
```

```python

# A column of name 'Label' is created in DataFrame
# Categorizing Age into 4 Categories
# Baby/Toddler: (0,3], 0 is excluded & 3 is included
# Child: (3,17], 3 is excluded & 17 is included
# Adult: (17,63], 17 is excluded & 63 is included
# Elderly: (63,99], 63 is excluded & 99 is included
df['Label'] = pd.cut(x=df['Age'], bins=[0, 3, 17, 63, 99],
                     labels=['Baby/Toddler', 'Child', 'Adult',
                             'Elderly'])

# Printing DataFrame after sorting Continuous to
# Categories
print(df)

# Check the number of values in each bin
print("Categories: ")
print(df['Label'].value_counts())
```

Input/Output



```
    Age         Label
0    42         Adult
1    15         Child
2    67       Elderly
3    55         Adult
4     1  Baby/Toddler
5    29         Adult
6    75       Elderly
7    89       Elderly
8     4         Child
9    10         Child
10   15         Child
11   38         Adult
12   22         Adult
13   77       Elderly
Categories:
Adult           5
Elderly         4
Child           4
Baby/Toddler    1
Name: Label, dtype: int64
```

Handling Categorical values

A categorical Feature can be either Nominal or Ordinal datatype.

- **Nominal:** Categories without any implied order. For example, different blood groups like A+ve, gender, fruits name etc.

- **Ordinal:** Categories with implied ordering. For example, rank, education level, grade etc.

See that there are also categorical values in the dataset, for this, you need to use Label Encoding or One Hot Encoding. To handling categorical data, we will learn 3 methods as following:

Label encoding

Label Encoding refers to converting the labels into a numeric form so as to convert them into the machine-readable form. Machine learning algorithms can then decide in a better way how those labels must be operated.

Implementation in Python

```
# Import libraries
import numpy as np
import pandas as pd

# Import dataset
df = pd.read_csv('../../data/Iris.csv')

print(df['species'].unique())


# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['species']= label_encoder.fit_transform(df['species'])

df['species'].unique()
```

Input/Output

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
array([0, 1, 2], dtype=int64)
```

Ordinal encoding

In ordinal encoding, each unique category value is assigned an integer value. For example, “red” is 1, “green” is 2, and “blue” is 3.

Implementation in Python

```
# example of a ordinal encoding
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define ordinal encoding
encoder = OrdinalEncoder()
# transform data
result = encoder.fit_transform(data)
print(result)
```

Input/Output

```
[['red'] ['green'] ['blue']]
[[2.] [1.] [0.]]
```

One-Hot Encoding

For categorical variables where no ordinal relationship exists, the integer encoding may not be enough, at best, or misleading to the model at worst. This is where the integer encoded variable is removed and one new binary variable is added for each unique integer value in the variable.

Implementation in Python

```python
# example of a one hot encoding
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```

Input/Output

```
[['red'] ['green'] ['blue']]
[[0. 0. 1.] [0. 1. 0.] [1. 0. 0.]]
```

2.2.2 Normalization

In machine learning, some feature values differ from others multiple times. The features with higher values will dominate the learning process. Therefore, data normalization could also be a typical practice in machine learning which consists of transforming numeric columns to a standard scale. We'll learn about 2 normalizing techniques given as follows:

The min-max feature scaling

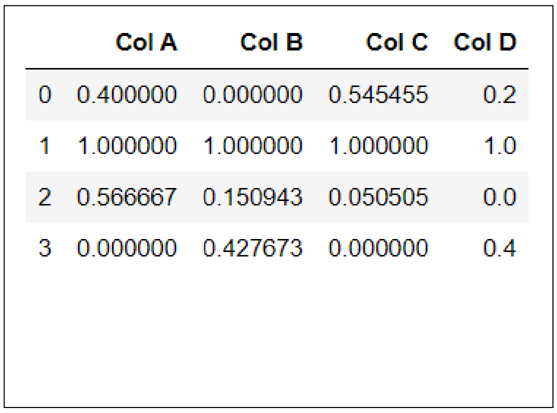
The min-max approach (often called normalization) rescales the feature to a hard and fast range of [0,1] by subtracting the minimum value of the feature then dividing by the range. We can apply the min-max scaling in Pandas using the .min() and .max() methods.

Implementation in Python

```python
import pandas as pd

# create data
df = pd.DataFrame([
                   [180000, 110, 18.9, 1400],
                   [360000, 905, 23.4, 1800],
                   [230000, 230, 14.0, 1300],
                   [60000, 450, 13.5, 1500]],

                   columns=['Col A', 'Col B',
                            'Col C', 'Col D'])
df_min_max_scaled = df.copy()

```

```python
# 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)
```

Input/Output



	Col A	Col B	Col C	Col D
0	0.400000	0.000000	0.545455	0.2
1	1.000000	1.000000	1.000000	1.0
2	0.566667	0.150943	0.050505	0.0
3	0.000000	0.427673	0.000000	0.4

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.

Python implementation: Try Yourself!!!

Lab Task (Please implement yourself and show the output to the instructor)

- Download and load a dataset. Now, write a Python program to impute null values (if any) using the average value of it's previous and next value.
- Using one-hot encoding, convert categorical values into numerical values.

3 Discussion & Conclusion

Based on the focused objective(s) to understand about the python program to process raw data, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

4 Lab Exercise (Submit as a report)

- Develop a jupyter notebook in Colab or Kaggle using all of the data processing criteria shown in the class. Choose an appropriate dataset from kaggle or any other sources containg NULL and garbage values (If not found, manipulate dataset by your own), then show the effect of all techniques with proper documentation in the notebook.

5 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected