ASSIGNMENT NO. 02

TITLE: DATA WRANGLING

PROBLEM STATEMENT: Create an "Academic performance" dataset of students and perform the following operations using Python.
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.
2. Scan all numeric variables for outliers. If there are outliers, use any of the suitable techniques to deal with them.
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.

OBJECTIVES:

- Apply steps to deal with missing values and outliers.
- Apply data transformation techniques to convert it into a usable format.

OUTCOMES:

- Students will be able to identify the missing data and outliers in the given data.
- Students will be able to apply appropriate techniques to deal with missing values and outliers in the given data.
- Students will be able to identify the need for data transformation and apply appropriate data transformation techniques.

SOFTWARE & HARDWARE REQUIREMENTS:
1. 64-bit Open source Linux or its derivative
2. Python, Jupiter Notebook, or Google Colab

THEORY:

Missing data is common in data science and machine learning areas.
Missing values can heavily influence your models, depending on how you handle them. The missing value may have significant meaning within your model and your target problem.
Types of Missing Values

- Missing completely at random (MCAR)

- Missing at random (MAR)
- Missing not at random (MNAR)
- Structurally Missing Data

Missing completely at random (MCAR)
These are the missing data points that follow no discernable pattern.
An example is data generated explicitly at random or survey data using a random subset of questions from a pre-defined list.
Missing at random (MAR)
In contrast to MCAR, MAR does not assume that the other variables cannot predict the missing value. This type is perhaps more often the case when there are errors in recording the data correctly.
For example, imagine a sensor that misses a particular minute's measurement but captures that data the minute before and the minute following.
Missing not at random (MNAR)
Missing not at random or nonignorable data is data where the mechanism for why the data is missing is known. Still, the values can not effectively be inferred or predicted.
An example of this form of data is a demographic for a survey avoiding questions. For example, perhaps people in a certain age/income bracket refuse to answer how many vehicles or houses they own.
Structurally Missing Data
The missing data is missing for an apparent reason.
This type is data that is missing on purpose.
For example, a survey that asks for income from employment would have missing values for those who do not have a job.

Checking for Missing Values

```
import pandas as pd
test=pd.read_csv("path to the csv file")
missing_values=test.isnull().sum()
print(missing_values)
```

Dealing with Missing Data

1. Ignore the missing values
Missing data under 10% for an individual case or observation can generally be ignored, except when the missing data is a MAR or MNAR.
The number of complete cases, i.e., observation with no missing data, must be sufficient for the selected analysis technique if the incomplete instances are not considered.
2. Drop the missing values

If the data is MCAR or MAR and the number of missing values in a feature is very high, that feature should be left out of the analysis. Suppose missing data for a particular feature or sample is more than 5%. In that case, you probably should leave that feature or sample out.
Suppose the cases or observations have missing target variables(s) values. It is advisable to delete the dependent variable(s) to avoid any artificial increase in relationships with independent variables.

3. Imputation

Deletion may not be the most effective option. For example, if too much information is discarded, it may not be possible to complete a reliable analysis. Or there may be insufficient data to generate a reliable prediction for observations that have missing data.
Instead of deletion, data scientists have multiple solutions to impute the value of missing data.
Imputation is the process of substituting the missing data with some statistical methods.
Imputation is useful because it preserves all cases by replacing missing data with an estimated value based on other available information.
But imputation methods should be used carefully as most of them introduce a large amount of bias and reduce variance in the dataset.

3.1 Imputation with mean

When a continuous variable column has missing values, you can calculate the mean of the non-null values and use it to fill the vacancies.

3.2 Imputation with Median

The missing values of a continuous feature can be filled with the median of the remaining non-null values. The median's advantage is that it is unaffected by the outliers, unlike the mean.

3.3 Missing values in categorical data

What if data is missing in the case of a categorical feature?
We can fill the missing values with the mode or most frequently occurring class/category.

```
#Imputing the Missing Value
import numpy as np
test['column_name']=test['column_name'].replace(np.NaN,test['column_name'].mean())

#Replacing With Arbitrary Value
test['column_name'] = test['column_name'].fillna(0)
test['column_name'].isnull().sum()

#Imputing Missing Values For Categorical Features using most frequent category
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='most_frequent')
imputer.fit_transform(X)

#Imputing Missing Values For Categorical Features using fixed category
imputer = SimpleImputer(strategy='constant', fill_value='missing')
```

```
imputer.fit_transform(X)
```

Outliers in Data

An Outlier is an observation in a given dataset that lies far from the rest of the observations. That means an outlier is vastly larger or smaller than the remaining values in the set. In statistics, we have three measures of central tendency, namely Mean, Median, and Mode. They help us describe the data. Mean is the accurate measure to describe the data when we do not have any outliers present.
Median is used if there is an outlier in the dataset.
Mode is used if there is an outlier AND about ½ or more of the data is the same.
'Mean' is the only measure of central tendency affected by the outliers, impacting Standard deviation.

Detecting Outliers

Below are some of the techniques of detecting outliers

- Boxplots
- Z-score
- Inter Quantile Range(IQR)

```
#Detecting outliers using Boxplot
import seaborn as sns
sns.boxplot(data=df[‘column_name’],x=df['column_name'])

#Detecting Outliers with Z-scores
import numpy as np
outliers = []
def detect_outliers_zscore(data):
    thres = 3
    mean = np.mean(data)
    std = np.std(data)
    # print(mean, std)
    for i in data:
        z_score = (i-mean)/std
        if (np.abs(z_score) > thres):
            outliers.append(i)
    return outliers
column_outliers = detect_outliers_zscore(column_name)
print("Outliers from Z-scores method: ",  column_outliers)

#Detecting outliers using the Inter Quantile Range(IQR)
outliers = []
def detect_outliers_iqr(data):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    # print(q1, q3)
    IQR = q3-q1
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)
    # print(lwr_bound, upr_bound)
    for i in data:
        if (i<lwr_bound or i>upr_bound):
            outliers.append(i)
    return outliers
column_outliers = detect_outliers_iqr(column_name)
print("Outliers from IQR method: ",  column_outliers)
```

Below are some of the methods of treating the outliers

- Trimming/removing the outlier
- Quantile based flooring and capping
- Mean/Median imputation

Data Transformation

Data transformation is the process of converting raw data into a format or structure that would be more suitable for the model or algorithm and data discovery in general. It is an essential step in feature engineering that facilitates discovery insights.
Why do we need data transformation?
All machine learning algorithms are based on mathematics. So, we need to convert all the columns into the numerical format. the algorithm is more likely to be biased when the data distribution is skewed transforming data into the same scale allows the algorithm to better compare the relative relationship between data points.
Most machine learning algorithms cannot operate directly with categorical data.
To address the problems associated with categorical data, we can use an encoding. This is how we convert a categorical variable into a numerical form.

Techniques for Encoding

- Replacing
- Label Encoding
- One-hot Encoding

Replacing :
This is a technique in which we replace the categorical data with a number. This is a simple replacement and does not involve much logical processing.

Label Encoding :

This is a technique in which we replace each value in a categorical column with numbers from 0 to N-1. Label encoding is the best method to use for ordinal data.
Predictive models that use this numerical data for analysis might sometimes mistake these labels for some order (for example, a model might think that a label of 3 is "better" than a label of 1, which is incorrect). In order to avoid this confusion, we can use a one-hot encoding.

```
#import the LabelEncoder class
from sklearn.preprocessing import LabelEncoder
#Creating the object instance
label_encoder = LabelEncoder()
for i in data_column_category:
    dataframe[i] = label_encoder.fit_transform( dataframe[i])
print("Label Encoded Data: ")
dataframe.head()
```

One-Hot Encoding :
In One-Hot Encoding, the label-encoded data is further divided into n number of columns.
Here, n denotes the total number of unique labels generated while performing label encoding.
For example, say that three new labels are generated through label encoding. Then, while performing one-hot encoding, the columns will be divided into three parts.

```
#Performing One Hot Encoding
from sklearn.preprocessing import OneHotEncoder
onehot_encoder = OneHotEncoder(sparse=False)
onehot_encoded = onehot_encoder.fit_transform(datafrmae[column_name])
```

CONCLUSION: To be written by students