**Assignment - 2**

. Perform the following operations using Python on the Air quality and Heart Diseases data sets

a. Data cleaning

* 1. b. Data integration
  2. c. Data transformation
  3. d. Error correcting
  4. e. Data model building

THEORY: Data cleaning Data cleansing or data cleaning is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

1. **Data cleansing** may be performed interactively with data wrangling tools, or as batch processing through scripting.

2. **Data integration** : Data integration involves combining data residing in different sources and providing users with a unified view of them. This process becomes significant in a variety of situations, which include both commercial (such as when two similar companies need to merge their databases) and scientific (combining research results from different bioinformatics repositories, for example) domains.

Data integration appears with increasing frequency as the volume and the need to share existing data explodes.[3] It has become the focus of extensive theoretical work, and numerous open problems remain unsolved.

**3. Data transformation** : In computing, data transformation is the process of converting data from one format or structure into another format or structure. It is a fundamental aspect of most data integration and data management tasks such as data wrangling, data warehousing, data integration and application integration.

**4. Error correcting** : Error detection and correction or error control are techniques that enable reliable delivery of digital data over unreliable communication channels. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. Error detection techniques allow detecting such errors, while error correction enables reconstruction of the original data in many cases.

5. **Data Modeling :** python has been the language of choice for predictive analysis due to its innumerable packages and strong developer community. Stages of Predictive Modeling Predictive Modeling is the process of building a model to predict future outcomes using statistics techniques. In order to generate the model, historical data of prior occurrences needs to be analyzed, classified and validated.

**a. Data cleaning**

[**Data cleaning or cleansing**](https://en.wikipedia.org/wiki/Data_cleansing) is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data.

*how to**find and clean*:

Missing Data –

* There are 4 ways to find the null values if present in the dataset.

**Using isnull() function (**data.isnull()) **:** This function provides the boolean value for the complete dataset to know if any null value is present or not.

**Using isna() function:data.isna() :** This is the same as the isnull() function. Ans provides the same output.

**Using isna().any()**

This function also gives a boolean value if any null value is present or not, but it gives results column-wise, not in tabular format.

**Using isna(). sum()**

This function gives the sum of the null values preset in the dataset column-wise.

**Using isna().any().sum()**

This function gives output in a single value if any null is present or not.

if there are any null values preset we can fill those places with any other value using the fillna() function of DataFrame.

Following is the syntax of fillna() function:

DataFrame\_name.fillna(*value=None*, *method=None*, *axis=None*, *inplace=False*, *limit=None*, *downcast=None*)

Cleaning / Filling Missing Data

Pandas provides various methods for cleaning the missing values. The fillna function can “fill in” NA values with non-null data in a couple of ways.

Replace NaN with a Scalar Value

The following program shows how you can replace "NaN" with "0".

df.fillna(0)

Drop Missing Values

If you want to simply exclude the missing values, then use the **dropna** function along with the **axis** argument. By default, axis=0, i.e., along row, which means that if any value within a row is NA then the whole row is excluded.

df.dropna()

In statistics, this method is called the listwise deletion technique. In this solution, we drop the entire observation as long as it contains a missing value.

*Only if* we are sure that the missing data is not informative, we perform this. Otherwise, we should consider other solutions.

df=df.dropna(subset=['date']) # dropping rows where no date is available

**Drop the Observation**

Here, we consider that only a minimal amount of observations have over 5 features missing altogether. We may create a new dataset df\_less\_missing\_rows deleting observations with over 5 missing features.

# drop rows with a lot of missing values

|  |
| --- |
| *ind\_missing=df[df['num\_missing'] >5].index* |
| *df\_less\_missing\_rows=df.drop(ind\_missing, axis=0)* |

**Drop the Feature**

we *only*do this when we are confident that this feature doesn’t provide useful information.

**Dropping of less valued columns:** stn\_code, agency, sampling\_date, location\_monitoring\_agency do not add much value to the dataset in terms of information. Therefore, we can drop those columns.

df=df.drop(['stn\_code', 'agency','sampling\_date','location\_monitoring\_station'], axis = 1)  #dropping columns that aren't required

**Changing the types to uniform format:**

When you see the dataset, you may notice that the ‘type’ column has values such as ‘Industrial Area’ and ‘Industrial Areas’ — both actually mean the same, so let’s remove such type of stuff and make it uniform.

**Notice that the ‘type’ column has values such as ‘Industrial Area’ and ‘Industrial Areas’ — both actually mean the same, so let’s remove them and make it uniform**

df["type"].unique()

types = {

    "Residential": "R",

    "Residential and others": "RO",

    "Residential, Rural and other Areas": "RRO",

    "Industrial Area": "I",

    "Industrial Areas": "I",

    "Industrial": "I",

    "Sensitive Area": "S",

    "Sensitive Areas": "S",

    "Sensitive": "S",

    "NaN": "RRO"

}

df.type = df.type.replace(types)

**Creating a year column**

To view the trend over a period of time, we need year values for each row and also when you see in most of the values in date column only has ‘year’ value. So, let’s create a new column holding year values.

# Creating a year column

**To view the trend over a period of time, we need year values for each row and also when you see in most of the values in date column only has ‘year’ value. So, let’s create a new column holding year values.**

df['date'] = pd.to\_datetime(df['date'], errors='coerce')

df.head(5)

df['year'] = df.date.dt.year

df.head(5)

# Handling Missing Values

**The column such as SO2, NO2, rspm, spm, pm2\_5 are the ones which contribute much to our analysis. So, we need to remove null from those columns to avoid inaccuracy in the prediction. We use the Imputer from sklearn.preprocessing to fill the missing values in every column with the mean.**

# defining columns of importance, which shall be used reguarly

COLS = ['so2', 'no2', 'rspm', 'spm', 'pm2\_5']

import numpy as np

from sklearn.impute import SimpleImputer

# invoking SimpleImputer to fill missing values

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')

df[COLS] = imputer.fit\_transform(df[COLS])

Data Integration

>sport=c("hockey","baseball","football")

> league=c("L1","L2","L3")

> trophy=c("SACH","SAU","YUV")

>trophies1=cbind(sport,league,trophy)

> trophies1 sport league trophy [1,] "hockey""L1""SACH" [2,] "baseball""L2""SAU" [3,] "football""L3""YUV" > trophies2=data.frame(sport=("Swiming"),league=("Lee"),trophy=("GAV"),stringsAsFactors = FALSE) > trophies=rbind(trophies1,trophies2) > trophies sport league trophy 1 hockey L1 SACH 2 baseball L2 SAU 3 football L3 YUV 4 Swiming Lee GAV

Data Transformation

**All machine learning algorithms are based on mathematics. So, we need to convert all the columns into numerical format.**

Taking a broader perspective, data is classified into numerical and categorical data:

1. Numerical: As the name suggests, this is numeric data that is quantifiable.
2. Categorical: The data is a string or non-numeric data that is qualitative in nature.

## Discovering Duplicates

Duplicate rows are rows that have been registered more than one time.

By taking a look at our test data set, we can assume that row 11 and 12 are duplicates.

To discover duplicates, we can use the duplicated() method.

The duplicated() method returns a Boolean values for each row:

Return True for every row that is a duplicate, otherwise False.

df.duplicated()

df.duplicated().sum()

## Removing Duplicates

To remove duplicates, use the drop\_duplicates() method.

df.drop\_duplicates(inplace = True)

**Simple Replacement of Categorical Data with a Number**

When we look at the categorical data, the first question that arises to anyone is how to handle those data, because machine learning is always good at dealing with numeric values. We could make machine learning models by using text data. So, to make predictive models we have to convert categorical data into numeric form.

1. Encoding: To address the problems associated with categorical data, we can use encoding.

This is the process by which we convert a categorical variable into a numerical form.

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

## Using replace() method

Replacing is one of the methods to convert categorical terms into numeric.

df.head()

df['type'].value\_counts()

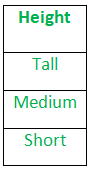
df['type'].replace({ 'MO':1, 'I':2, 's':3 , 'RO':4, 'K':5, 'RIRUO':6 }, inplace=True)

df.info()

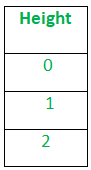
df['type']

**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. It is an important pre-processing step for the structured dataset in supervised learning.

**Example :**   
Suppose we have a column*Height* in some dataset.



After applying label encoding, the Height column is converted into:



where 0 is the label for tall, 1 is the label for medium, and 2 is a label for short height.

Exam. from sklearn.preprocessing import LabelEncoder

# label\_encoder object knows how to understand word labels.

labelencoder = LabelEncoder()

# Encode labels in column 'state'.

df['state'] =labelencoder.fit\_transform(df['state'])

**Limitation of label Encoding**

Label encoding converts the data in machine-readable form, but it assigns a unique number(starting from 0) to each class of data. This may lead to the generation of priority issues in the training of data sets. A label with a high value may be considered to have high priority than a label having a lower value.

**One-Hot Encoder**

# One-hot encoding is used to convert categorical variables into a format that can be readily used by machine learning algorithms.

Though label encoding is straight but it has the disadvantage that the numeric values can be misinterpreted by algorithms as having some sort of hierarchy/order in them. This ordering issue is addressed in another common alternative approach called ‘One-Hot Encoding’. In this strategy, each category value is converted into a new column and assigned a 1 or 0 (notation for true/false) value to the column.

**Using sci-kit learn library approach:**

OneHotEncoder from SciKit library only takes numerical categorical values, hence any value of string type should be label encoded before one hot encoded.

dfAndhra = df[df['state']==0]

dfAndhra

dfAndhra['location'].value\_counts()

from sklearn.preprocessing import OneHotEncoder

# creating instance of one-hot-encoder

onehotencoder = OneHotEncoder(sparse=False, handle\_unknown='error', drop='first')

#passing dfAndhra column (label encoded values of dfAndhra)

pd.DataFrame(onehotencoder.fit\_transform(dfAndhra[['location']]))

dfAndhra['location'].value\_counts()

**Error correction**

df.isnull().sum()

df=df.fillna(df.median())

df.isnull().sum()

**# Detecting and Filtering Outliers**

df.describe()

df[df['so2']>100]=0

**Detect Outliers**

Outliers are numerical values that lie significantly outside of the statistical norm. Cutting that down from unnecessary science garble – they are data points that are so out of range they are likely misreads. They, like duplicates, need to be removed.

**How to find out?**

Depending on whether the feature is numeric or categorical, we can use different techniques to study its distribution to detect outliers.

* **Technique #1:**[**Histogram**](https://en.wikipedia.org/wiki/Histogram)**/**[**Box Plot**](https://www.khanacademy.org/math/statistics-probability/summarizing-quantitative-data/box-whisker-plots/a/box-plot-review)

When the feature is numeric, we can use a histogram and box plot to detect outliers.

* **Technique #2: Descriptive Statistics**

Also, for numeric features, the outliers could be too distinct that the box plot can’t visualize them. Instead, we can look at their descriptive statistics.

For example, for the feature *life\_sq*again, we can see that the maximum value is 7478, while the 75% quartile is only 43. The 7478 value is an outlier.

**Data Model building : (write answer for following in journal)**

Explain SVM

Explain performance measure like precision, recall , Accuracy

Explain Data model building

FAQ :

* How data analytics performed using python?

• What is the need of data analytics?

• Explain different types of data analytics.

• Why data transformation is important in Big Data Analytics?

• Explain data cleaning process.

• Why data is dirty?

• Explain data analytics life cycle.

• What is data analytics?