Assignment No - 01 (Group A)

Problem statement:

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.

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.

4. Data Pre-processing: 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.

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.

6. Turn categorical variables into quantitative variables in Python. 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.

Pre-requisite

- Python programing concepts

Objective

1. Should be able to make raw data useful by applying scientific data processing libraries.

2. Understand and apply the python libraries such as numpy, pandas, matplotlib etc.

Software and Hardware requirements:-

1. **Operating system:** Linux- Ubuntu 16.04 to 17.10, or Windows 7 to 10,
2. **RAM-** 2GB RAM (4GB preferable)
3. IDE :-Anaconda Jupiter Notebook / pycharm /Visual Studio

Theory-

Data wrangling

Data wrangling referred to as data mining, is the process of transforming and mapping data from one raw data into another format with the intent of making it more appropriate and valuable for variety of downstream purpose. Such as analytic.

The Goal of Data wrangling is to assure quality and useful data.

Data analytic typically spend the majority of their time in process of Data wrangling compared to the actual analysis of data.

The process of Data wrangling may include further mining, data visualization, data aggregation, training a statistical model, as well as many other uses.

Data wrangling typically follows a set of general steps which begins with extracting the data in raw form from the data source, parsing the data into predefined data structures and finding depositing the result content into a data link for storage and future use.

Methods and function used:-

1. import panda as pd:-

pd.read_csv():-

CSV files are the Comma Separated Files. It allows users to load tabular data into a DataFrame, which is a powerful structure for data manipulation and analysis.

To access data from the CSV file, we require a function read_csv() from Pandas that retrieves data in the form of the data frame.

Syntax: pd.read_csv(filepath_or_buffer,sep=',’header=’infer’, index_col=None, usecols=None, engine=None, skiprows=None, nrows=None)

Parameters:

filepath_or_buffer: Location of the csv file. It accepts any string path or URL of the file.

sep: It stands for separator, default is ‘, ‘.

header: It accepts int, a list of int, row numbers to use as the column names, and the start of the data. If no names are passed, i.e., header=None, then, it will display the first column as 0, the second as 1, and so on.

usecols: Retrieves only selected columns from the CSV file.

nrows: Number of rows to be displayed from the dataset.

index_col: If None, there are no index numbers displayed along with records.

skiprows: Skips passed rows in the new data frame.

2. data_frame.head()

Return the first 5 rows of the DataFrame. The head() method returns the first 5 rows if a number is not specified.

Syntax dataframe.head(n)

A DataFrame with headers and the specified number of rows.

3. dataframe.shape

Return a tuple representing the dimensionality of the Data Frame.

Ex. An output(48,14) representing 48 rows and 14 columns

4. Conversion with convert_dtypes

The convert_dtypes DataFrame method will convert all columns to the best possible types that support pd.NA, the Pandas missing value.

Note that this method will not convert numeric strings to integers or floats.

Change all columns to the appropriate types

```
    df.convert_dtypes()
# Change column B and C's values to a numeric type
df[['B', 'C']] = df[['B', 'C']].apply(pd.to_numeric)
# Change column B and C's values to integers
df = df.astype({'B': int, 'C': int})
```

5. replace() method

Replacing is one of the methods to convert categorical terms into numeric. For example, We will take a dataset of people's salaries based on their level of education. This is an ordinal type of categorical variable. We will convert their education levels into numeric terms.

Syntax:

```
replace(to_replace=None, value=None, inplace=False, limit=None, regex=False, method='pad')
df['Education'].replace(['Under-Graduate', 'Diploma '],  [0, 1], inplace=True)
```

6.Dataframe. describe()

The describe() method returns description of the data in the DataFrame.

If the DataFrame contains numerical data, the description contains these information for each column:

count - The number of not-empty values.

mean - The average (mean) value.

std - The standard deviation.

min - the minimum value.

25% - The 25% percentile*.

50% - The 50% percentile*.

75% - The 75% percentile*.

max - the maximum value.

*Percentile meaning: how many of the values are less than the given percentile

7.isnull()

Replace all values in the DataFrame with True for NULL values, otherwise False:

Syntax

dataframe.isnull()

8 df.shape # check out the dimension of the dataset

9.df.dtypes # Look at the data types for each column

Do these match the expected data types for the columns?

10.df.head() # read the first five rows

11.df.tail() # read the Last five rows

12.df.columns.values # return an array of column names

13.df.columns.values.tolist() # return a List of column names

Check Missing Values

df.isnull() # Checking missing **Values**

df.notnull() # Checking non-missing **Values**

df.isnull().values.any() # Check if there's missing **Values**

df.isnull().sum().sum() # Check how many missing **Values** in data

df.isnull().sum() #Check missing by variable

df[df["height"].notnull()] #return raws where height is not null

Get Information about Missing Values

We could subset the data based on the missing values and create a new data frame to hold all the rows.

no_ missing =df.dropna()# if drop missing values and assign the data to no missing

no_ missing = no missing.reset_index(drop.True) #this resets the index bock to 0

You can also set a threshold of missing values. In the below example it drops rows that contain less than 50 non-missing values.

Threshold_missing = df.dropna(thresh=50)

If we use **dataframe.dropna(thresh=25)** to drop rows that contain less than 25 non-missing values, we don't change the original data. We can assign the output to a new variable or save the changes to the original data right away by using **dataframe.dropna(thresh=25, inplace=True)**. For our example, it would be **df.dropna(thresh=25, inplace=True)**

Fill In Missing Values

For quantitative variables, we may replace missing values with the sample mean, mode, median, or other numbers. For categorical variables, we can create a new category for missing values by replacing missing values with a string.

Replace missing values with 0.

Fill_no = df.fillna(0) # **Fill** in missing with 0 and save the data to Fill_no

df['DataFrame Column'] = df[‘DataFrame Column’].fillna(0) # Fill in missing for a singular column

Fill_no.head()

Replace all missing with string missing.

df.fillna("missing")

#**Fill** in missing with a string: -missing- and save the data to **Fill_str**

df[“Height”].fillna(df[“Height”].mean(), inplace=True)

# **Fill** missing values with the sample mean

Dropping Data

We may want to drop duplicate rows if any and save the changes to the original data.

df.drop_duplicates(inplace=True)

We also may want to drop some observations or some columns.

df = df.drop(df.index([1,2,3]]) # Droa the first second and third row

df = df.drop(df.index[range(1,11)]) # drop the 2nd to the 16th row

df = df.drop([“Height”], axis = 1) # drop the Height column from the data

df = df.drop([[“Height”, “Season”]], axis **=1)** #Drop the height and season columns from the dotoset

Sub setting

iloc stands for integer location. It helps subset data by using integers. It’s counterpart loc uses strings to find data within your data set.

df.iloc[0] # Show the first row of information

df.iloc[[0, 1, 2]] # show the first second and third row df.iloc[:,0] #print the first column

df.iloc[:,0:5] #print the first 5 columns

df.iloc[0:5,0:3] # a subset of the first 3 columns and 5 rows

df.loc[0] #Show the first row

df[[“Height", "Name", "Age“]] # Subset of height nome and age df.sample(n=100) # Random sample of 100 rows

df.sample(frac=0.1, replace=True) # A random sample of 10% of the data with replacement

Conclusion:

Data wrangling is one of the most important technique to turn raw data into useful asset where pandas perform an very important role of converting so called raw data into productive data set that can be utilized later for different purposes.