**Experiment No.1**

**Aim:** To understand the process of data preparation using NumPy and Pandas

**CO Mapped:**

**CO1:** To apply the process of data preparation for the given dataset to solve real-world problems

**Prerequisites:** Python3, basic syntax of NumPy and Pandas

**Problem Statement:**

Data preparation is the process of preparing raw data so that it is suitable for further processing and analysis.

Data preparation using NumPy and Pandas

a. Derive an index field and add it to the data set.

# b. Find out the missing values.

# c. Obtain a listing of all records that are outliers according to the any field.

i. Print out a listing of the 10 largest values for that field.

ii. Identify how many outliers there are and identify the most extreme outlier.

**Code:**

**import** pandas **as** pd

# What is BAD DATA

# Classification

# 1. Missing Values

# 2. Bad Values

# 3. Duplicated Data

In [22]:

df **=** pd**.**read\_csv('cart.csv')

df

Out[22]:

|  | **date** | **name** | **amount** | **price** |
| --- | --- | --- | --- | --- |
| **0** | 2021-03-01 | carrot | 7.0 | 5.73 |
| **1** | 2021-03-01 | egg | 12.0 | 1.70 |
| **2** | 2021-03-01 | milk | NaN | 3.57 |
| **3** | 2021-03-01 | potato | 2.0 | NaN |
| **4** | NaN | tomato | 6.0 | 1.52 |
| **5** | 2021-03-02 | potato | 3.0 | 2.17 |
| **6** | 2021-03-03 | NaN | 5.0 | 3.68 |

In [23]:

df **=** pd**.**read\_csv('cart.csv', parse\_dates**=**['date'])

df

Out[23]:

|  | **date** | **name** | **amount** | **price** |
| --- | --- | --- | --- | --- |
| **0** | 2021-03-01 | carrot | 7.0 | 5.73 |
| **1** | 2021-03-01 | egg | 12.0 | 1.70 |
| **2** | 2021-03-01 | milk | NaN | 3.57 |
| **3** | 2021-03-01 | potato | 2.0 | NaN |
| **4** | NaT | tomato | 6.0 | 1.52 |
| **5** | 2021-03-02 | potato | 3.0 | 2.17 |
| **6** | 2021-03-03 | NaN | 5.0 | 3.68 |

In [4]:

*#missing values*

df['amount']**.**astype('Int32')

Out[4]:

0 7

1 12

2 <NA>

3 2

4 6

5 3

6 5

Name: amount, dtype: Int32

In [5]:

df**.**isnull()

Out[5]:

|  | **date** | **name** | **amount** | **price** |
| --- | --- | --- | --- | --- |
| **0** | False | False | False | False |
| **1** | False | False | False | False |
| **2** | False | False | True | False |
| **3** | False | False | False | True |
| **4** | True | False | False | False |
| **5** | False | False | False | False |
| **6** | False | True | False | False |

In [6]:

df**.**isnull()**.**any(axis**=**1)

Out[6]:

0 False

1 False

2 True

3 True

4 True

5 False

6 True

dtype: bool

In [24]:

df**.**isnull()**.**any(axis**=**0)

Out[24]:

date True

name True

amount True

price True

dtype: bool

In [7]:

*# bad values*

*# %%*

df **=** pd**.**read\_csv('metrics.csv', parse\_dates**=**['time'])

df**.**sample(10)

Out[7]:

|  | **time** | **name** | **value** |
| --- | --- | --- | --- |
| **38** | 2021-07-13 14:37:10.563 | cpu | 2.737000e+01 |
| **83** | 2021-07-13 14:37:31.617 | mem | 2.275070e+08 |
| **13** | 2021-07-13 14:36:58.122 | cpu | 3.233000e+01 |
| **96** | 2021-07-13 14:37:38.316 | cpu | 3.507000e+01 |
| **24** | 2021-07-13 14:37:03.864 | cpu | 3.049000e+01 |
| **68** | 2021-07-13 14:37:24.918 | cpu | 3.337000e+01 |
| **12** | 2021-07-13 14:36:58.122 | mem | 2.275156e+08 |
| **55** | 2021-07-13 14:37:18.219 | mem | 2.275523e+08 |
| **7** | 2021-07-13 14:36:55.251 | cpu | 3.188000e+01 |
| **89** | 2021-07-13 14:37:34.488 | cpu | 3.946000e+01 |

In [8]:

*# %%*

df**.**groupby('name')**.**describe()

Out[8]:

|  | **value** | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **count** | **mean** | **std** | **min** | **25%** | **50%** | **75%** | **max** |
| **name** |  |  |  |  |  |  |  |  |
| **CPU** | 1.0 | 3.040000e+01 | NaN | 30.40 | 3.040000e+01 | 3.040000e+01 | 3.040000e+01 | 30.4 |
| **cpu** | 49.0 | 3.645633e+01 | 3.977927e+01 | -32.14 | 2.956000e+01 | 3.157000e+01 | 3.507000e+01 | 300.9 |
| **mem** | 50.0 | 2.229711e+08 | 3.217637e+07 | 295.00 | 2.275070e+08 | 2.275188e+08 | 2.275342e+08 | 227556054.0 |

In [9]:

*# %%*

df['name']**.**value\_counts()

Out[9]:

mem 50

cpu 49

CPU 1

Name: name, dtype: int64

In [17]:

*#Duplicate Value*

*# %%*

*# %%*

df **=** pd**.**read\_csv('cart-dup.csv', parse\_dates**=**['date'])

df

Out[17]:

|  | **date** | **name** | **amount** | **price** |
| --- | --- | --- | --- | --- |
| **0** | 2021-03-01 | carrot | 7 | 5.73 |
| **1** | 2021-03-01 | egg | 12 | 1.70 |
| **2** | 2021-03-01 | egg | 12 | 1.20 |
| **3** | 2021-03-01 | milk | 1 | 3.57 |
| **4** | 2021-03-02 | potato | 3 | 2.17 |
| **5** | 2021-03-02 | potato | 3 | 2.17 |

In [18]:

*# %% consider whole row*

df**.**duplicated()

Out[18]:

0 False

1 False

2 False

3 False

4 False

5 True

dtype: bool

In [19]:

*# %%*

df**.**duplicated(['date', 'name'])

Out[19]:

0 False

1 False

2 True

3 False

4 False

5 True

dtype: bool

# a. Derive an index field and add it to the data set.

Index is like an address, that’s how any data point across the dataframe or series can be accessed. Rows and columns both have indexes, rows indices are called as index and for columns its general column names.

DataFrame.set\_index(keys, drop=True, append=False, inplace=False, verify\_integrity=False)

keys: Column name or list of column name. drop: Boolean value which drops the column used for index if True. append: Appends the column to existing index column if True. inplace: Makes the changes in the dataframe if True. verify\_integrity: Checks the new index column for duplicates if True.

In [26]:

*# making data frame from csv file*

data **=** pd**.**read\_csv("employees.csv")

*# display*

data**.**head()

Out[26]:

|  | **First Name** | **Gender** | **Start Date** | **Last Login Time** | **Salary** | **Bonus %** | **Senior Management** | **Team** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Douglas | Male | 8/6/1993 | 12:42 PM | 97308 | 6.945 | True | Marketing |
| **1** | Thomas | Male | 3/31/1996 | 6:53 AM | 61933 | 4.170 | True | NaN |
| **2** | Maria | Female | 4/23/1993 | 11:17 AM | 130590 | 11.858 | False | Finance |
| **3** | Jerry | Male | 3/4/2005 | 1:00 PM | 138705 | 9.340 | True | Finance |
| **4** | Larry | Male | 1/24/1998 | 4:47 PM | 101004 | 1.389 | True | Client Services |

Code #1: Changing Index column In this example, First Name column has been made the index column of Data Frame.

In [27]:

*# First Name column has been made the index column of Data Frame.*

*# setting first name as index column*

data**.**set\_index("First Name", inplace **=** **True**)

*# display*

data**.**head()

Out[27]:

|  | **Gender** | **Start Date** | **Last Login Time** | **Salary** | **Bonus %** | **Senior Management** | **Team** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **First Name** |  |  |  |  |  |  |  |
| **Douglas** | Male | 8/6/1993 | 12:42 PM | 97308 | 6.945 | True | Marketing |
| **Thomas** | Male | 3/31/1996 | 6:53 AM | 61933 | 4.170 | True | NaN |
| **Maria** | Female | 4/23/1993 | 11:17 AM | 130590 | 11.858 | False | Finance |
| **Jerry** | Male | 3/4/2005 | 1:00 PM | 138705 | 9.340 | True | Finance |
| **Larry** | Male | 1/24/1998 | 4:47 PM | 101004 | 1.389 | True | Client Services |

In [36]:

*# making data frame from csv file*

data **=** pd**.**read\_csv("employees.csv")

*# setting first name as index column*

data**.**set\_index(["First Name", "Gender"], inplace **=** **True**,

append **=** **True**, drop **=** **False**)

*# display*

data**.**head()

Out[36]:

|  |  |  | **First Name** | **Gender** | **Start Date** | **Last Login Time** | **Salary** | **Bonus %** | **Senior Management** | **Team** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **First Name** | **Gender** |  |  |  |  |  |  |  |  |
| **0** | **Douglas** | **Male** | Douglas | Male | 8/6/1993 | 12:42 PM | 97308 | 6.945 | True | Marketing |
| **1** | **Thomas** | **Male** | Thomas | Male | 3/31/1996 | 6:53 AM | 61933 | 4.170 | True | NaN |
| **2** | **Maria** | **Female** | Maria | Female | 4/23/1993 | 11:17 AM | 130590 | 11.858 | False | Finance |
| **3** | **Jerry** | **Male** | Jerry | Male | 3/4/2005 | 1:00 PM | 138705 | 9.340 | True | Finance |
| **4** | **Larry** | **Male** | Larry | Male | 1/24/1998 | 4:47 PM | 101004 | 1.389 | True | Client Services |

In [37]:

*# importing pandas library*

**import** pandas **as** pd

*# creating and initializing a nested list*

students **=** [['jack', 34, 'Sydeny', 'Australia',85.96],

['Riti', 30, 'Delhi', 'India',95.20],

['Vansh', 31, 'Delhi', 'India',85.25],

['Nanyu', 32, 'Tokyo', 'Japan',74.21],

['Maychan', 16, 'New York', 'US',99.63],

['Mike', 17, 'las vegas', 'US',47.28]]

*# Create a DataFrame object*

df **=** pd**.**DataFrame(students,

columns**=**['Name', 'Age', 'City', 'Country','Agg\_Marks'],

index**=**['a', 'b', 'c', 'd', 'e', 'f'])

*# here we set Float column 'Agg\_Marks' as index of data frame*

*# using dataframe.set\_index() function*

df **=** df**.**set\_index('Agg\_Marks')

*# Displaying the Data frame*

df

Out[37]:

|  | **Name** | **Age** | **City** | **Country** |
| --- | --- | --- | --- | --- |
| **Agg\_Marks** |  |  |  |  |
| **85.96** | jack | 34 | Sydeny | Australia |
| **95.20** | Riti | 30 | Delhi | India |
| **85.25** | Vansh | 31 | Delhi | India |
| **74.21** | Nanyu | 32 | Tokyo | Japan |
| **99.63** | Maychan | 16 | New York | US |
| **47.28** | Mike | 17 | las vegas | US |

In [38]:

*# importing pandas library*

**import** pandas **as** pd

*# creating and initializing a nested list*

students **=** [['jack', 34, 'Sydeny', 'Australia',85.96,400],

['Riti', 30, 'Delhi', 'India',95.20,750],

['Vansh', 31, 'Delhi', 'India',85.25,101],

['Nanyu', 32, 'Tokyo', 'Japan',74.21,900],

['Maychan', 16, 'New York', 'US',99.63,420],

['Mike', 17, 'las vegas', 'US',47.28,555]]

*# Create a DataFrame object*

df **=** pd**.**DataFrame(students,

columns**=**['Name', 'Age', 'City', 'Country','Agg\_Marks','ID'],

index**=**['a', 'b', 'c', 'd', 'e', 'f'])

*# Here we pass list of 3 columns i.e 'Name', 'City' and 'ID'*

*# to dataframe.set\_index() function*

*# to set them as multiIndex of dataframe*

df **=** df**.**set\_index(['Name','City','ID'])

*# Displaying the Data frame*

df

Out[38]:

|  |  |  | **Age** | **Country** | **Agg\_Marks** |
| --- | --- | --- | --- | --- | --- |
| **Name** | **City** | **ID** |  |  |  |
| **jack** | **Sydeny** | **400** | 34 | Australia | 85.96 |
| **Riti** | **Delhi** | **750** | 30 | India | 95.20 |
| **Vansh** | **Delhi** | **101** | 31 | India | 85.25 |
| **Nanyu** | **Tokyo** | **900** | 32 | Japan | 74.21 |
| **Maychan** | **New York** | **420** | 16 | US | 99.63 |
| **Mike** | **las vegas** | **555** | 17 | US | 47.28 |

# b. Find out the missing values.

In [39]:

df **=** pd**.**read\_csv('ships.csv')

df

*# %%*

df**.**dtypes

Out[39]:

name object

lat float64

lng float64

dtype: object

In [40]:

*# importing pandas as pd*

**import** pandas **as** pd

*# importing numpy as np*

**import** numpy **as** np

*# dictionary of lists*

dict **=** {'First Score':[100, 90, np**.**nan, 95],

'Second Score': [30, 45, 56, np**.**nan],

'Third Score':[np**.**nan, 40, 80, 98]}

*# creating a dataframe from list*

df **=** pd**.**DataFrame(dict)

*# using isnull() function*

df**.**isnull()

Out[40]:

|  | **First Score** | **Second Score** | **Third Score** |
| --- | --- | --- | --- |
| **0** | False | False | True |
| **1** | False | False | False |
| **2** | True | False | False |
| **3** | False | True | False |

In [41]:

*# importing pandas as pd*

**import** pandas **as** pd

*# importing numpy as np*

**import** numpy **as** np

*# dictionary of lists*

dict **=** {'First Score':[100, 90, np**.**nan, 95],

'Second Score': [30, 45, 56, np**.**nan],

'Third Score':[np**.**nan, 40, 80, 98]}

*# creating a dataframe using dictionary*

df **=** pd**.**DataFrame(dict)

*# using notnull() function*

df**.**notnull()

Out[41]:

|  | **First Score** | **Second Score** | **Third Score** |
| --- | --- | --- | --- |
| **0** | True | True | False |
| **1** | True | True | True |
| **2** | False | True | True |
| **3** | True | False | True |

# Handling missing values

In [42]:

*# dictionary of lists*

dict **=** {'First Score':[100, 90, np**.**nan, 95],

'Second Score': [30, 45, 56, np**.**nan],

'Third Score':[np**.**nan, 40, 80, 98]}

*# creating a dataframe from dictionary*

df **=** pd**.**DataFrame(dict)

*# filling missing value using fillna()*

df**.**fillna(0)

Out[42]:

|  | **First Score** | **Second Score** | **Third Score** |
| --- | --- | --- | --- |
| **0** | 100.0 | 30.0 | 0.0 |
| **1** | 90.0 | 45.0 | 40.0 |
| **2** | 0.0 | 56.0 | 80.0 |
| **3** | 95.0 | 0.0 | 98.0 |

In [43]:

*# dictionary of lists*

dict **=** {'First Score':[100, 90, np**.**nan, 95],

'Second Score': [30, 45, 56, np**.**nan],

'Third Score':[np**.**nan, 40, 80, 98]}

*# creating a dataframe from dictionary*

df **=** pd**.**DataFrame(dict)

*# filling a missing value with*

*# previous ones*

df**.**fillna(method **=**'pad')

Out[43]:

|  | **First Score** | **Second Score** | **Third Score** |
| --- | --- | --- | --- |
| **0** | 100.0 | 30.0 | NaN |
| **1** | 90.0 | 45.0 | 40.0 |
| **2** | 90.0 | 56.0 | 80.0 |
| **3** | 95.0 | 56.0 | 98.0 |

In [44]:

*# dictionary of lists*

dict **=** {'First Score':[100, 90, np**.**nan, 95],

'Second Score': [30, 45, 56, np**.**nan],

'Third Score':[np**.**nan, 40, 80, 98]}

*# creating a dataframe from dictionary*

df **=** pd**.**DataFrame(dict)

*# filling null value using*

*# # next ones*

df**.**fillna(method **=**'bfill')

Out[44]:

|  | **First Score** | **Second Score** | **Third Score** |
| --- | --- | --- | --- |
| **0** | 100.0 | 30.0 | 40.0 |
| **1** | 90.0 | 45.0 | 40.0 |
| **2** | 95.0 | 56.0 | 80.0 |
| **3** | 95.0 | NaN | 98.0 |

In [45]:

*# making data frame from csv file*

data **=** pd**.**read\_csv("employees.csv")

*# will replace Nan value in dataframe with value -99*

data**.**replace(to\_replace **=** np**.**nan, value **=** **-**99)

Out[45]:

|  | **First Name** | **Gender** | **Start Date** | **Last Login Time** | **Salary** | **Bonus %** | **Senior Management** | **Team** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Douglas | Male | 8/6/1993 | 12:42 PM | 97308 | 6.945 | True | Marketing |
| **1** | Thomas | Male | 3/31/1996 | 6:53 AM | 61933 | 4.170 | True | -99 |
| **2** | Maria | Female | 4/23/1993 | 11:17 AM | 130590 | 11.858 | False | Finance |
| **3** | Jerry | Male | 3/4/2005 | 1:00 PM | 138705 | 9.340 | True | Finance |
| **4** | Larry | Male | 1/24/1998 | 4:47 PM | 101004 | 1.389 | True | Client Services |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **995** | Henry | -99 | 11/23/2014 | 6:09 AM | 132483 | 16.655 | False | Distribution |
| **996** | Phillip | Male | 1/31/1984 | 6:30 AM | 42392 | 19.675 | False | Finance |
| **997** | Russell | Male | 5/20/2013 | 12:39 PM | 96914 | 1.421 | False | Product |
| **998** | Larry | Male | 4/20/2013 | 4:45 PM | 60500 | 11.985 | False | Business Development |
| **999** | Albert | Male | 5/15/2012 | 6:24 PM | 129949 | 10.169 | True | Sales |

1000 rows × 8 columns

In [48]:

*# dictionary of lists*

dict **=** {'First Score':[100, 90, np**.**nan, 95],

'Second Score': [30, np**.**nan, 45, 56],

'Third Score':[52, 40, 80, 98],

'Fourth Score':[np**.**nan, np**.**nan, np**.**nan, 65]}

*# creating a dataframe from dictionary*

df **=** pd**.**DataFrame(dict)

df

Out[48]:

|  | **First Score** | **Second Score** | **Third Score** | **Fourth Score** |
| --- | --- | --- | --- | --- |
| **0** | 100.0 | 30.0 | 52 | NaN |
| **1** | 90.0 | NaN | 40 | NaN |
| **2** | NaN | 45.0 | 80 | NaN |
| **3** | 95.0 | 56.0 | 98 | 65.0 |

In [49]:

*# using dropna() function*

df**.**dropna()

Out[49]:

|  | **First Score** | **Second Score** | **Third Score** | **Fourth Score** |
| --- | --- | --- | --- | --- |
| **3** | 95.0 | 56.0 | 98 | 65.0 |

# c. Obtain a listing of all records that are outliers according to the any field. i. Print out a listing of the 10 largest values for that field. ii. Identify how many outliers there are and identify the most extreme outlier.

In [56]:

*#load the data into a dataframe*

df **=** pd**.**read\_csv("uber.csv")

*#drop the unnecessary columns*

df **=** df**.**drop(columns**=**(["pickup\_longitude", "pickup\_latitude", "dropoff\_longitude", "dropoff\_latitude"]))

*#check the first 5 rows*

df**.**head()

Out[56]:

|  | **Unnamed: 0** | **key** | **fare\_amount** | **pickup\_datetime** | **passenger\_count** |
| --- | --- | --- | --- | --- | --- |
| **0** | 24238194 | 2015-05-07 19:52:06.0000003 | 7.5 | 2015-05-07 19:52:06 UTC | 1 |
| **1** | 27835199 | 2009-07-17 20:04:56.0000002 | 7.7 | 2009-07-17 20:04:56 UTC | 1 |
| **2** | 44984355 | 2009-08-24 21:45:00.00000061 | 12.9 | 2009-08-24 21:45:00 UTC | 1 |
| **3** | 25894730 | 2009-06-26 08:22:21.0000001 | 5.3 | 2009-06-26 08:22:21 UTC | 3 |
| **4** | 17610152 | 2014-08-28 17:47:00.000000188 | 16.0 | 2014-08-28 17:47:00 UTC | 5 |

# Since the data doesn’t follow a normal distribution, we will calculate the outlier data points using the statistical method called interquartile range (IQR) instead of using Z-score. Using the IQR, the outlier data points are the ones falling below Q1–1.5 IQR or above Q3 + 1.5 IQR. The Q1 is the 25th percentile and Q3 is the 75th percentile of the dataset, and IQR represents the interquartile range calculated by Q3 minus Q1 (Q3–Q1).

Using the convenient pandas .quantile() function, we can create a simple Python

In [64]:

*#create a function to find outliers using IQR*

**def** find\_outliers\_IQR(df):

q1**=**df**.**quantile(0.25)

q3**=**df**.**quantile(0.75)

IQR**=**q3**-**q1

outliers **=** df[((df**<**(q1**-**1.5**\***IQR)) **|** (df**>**(q3**+**1.5**\***IQR)))]

**return** outliers

outliers **=** find\_outliers\_IQR(df['fare\_amount'])

print('number of outliers: '**+** str(len(outliers)))

print('max outlier value: '**+** str(outliers**.**max()))

print('min outlier value: '**+** str(outliers**.**min()))

outliers

number of outliers: 17167

max outlier value: 499.0

min outlier value: -52.0

Out[64]:

6 24.50

30 25.70

34 39.50

39 29.00

48 56.80

...

199976 49.70

199977 43.50

199982 57.33

199985 24.00

199997 30.90

Name: fare\_amount, Length: 17167, dtype: float64

In [ ]:

**Conclusion: -**

In this experiment, we studied how using pandas and NumPy library we can pre-process the data.