LAB Manual

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No.08**

**A.1 Aim:** Perform Exploratory Data Analysis on real world dataset using Pandas library

* 1. Read different types of data files (text, csv, tsv, excel file etc.)
  2. Create data frames from data structures (dictionary, list, tuple)
  3. Create data frames from unformatted text file.
  4. Obtain metadata of given dataset
  5. Handling Missing Values in dataset
  6. Exploratory data analysis using different commands

**A.2 Prerequisite:**

Python Programming.

Students should go through the prerequisite documents if any, provided by faculty before attending lab.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to:**

1. View the metadata of given dataset.
2. Identify null and missing values. Replace them by appropriate values.
3. Perform exploratory data analysis using Pandas library.

**A.4 Theory:**

**In this section we will deal with basic commands in pandas used for exploratory data analysis.** Students need to read the document carefully. Students are expected to explore the commands given in between the documents for self-study using pandas documentation.. These commands are which are not described in this section. All examples and descriptions of the commands are taken from Pandas documentation website **(https://pandas.pydata.org/docs/).**

**Pandas** consists of the following things

* A set of labeled array data structures, the primary of which are Series/TimeSeries and DataFrame
* Index objects enabling both simple axis indexing and multi-level / hierarchical axis indexing
* An integrated group by engine for aggregating and transforming data sets
* Date range generation (date\_range) and custom date offsets enabling the implementation of customized frequencies
* Input/Output tools: loading tabular data from flat files (CSV, delimited, Excel 2003), and saving and loading pandas objects from the fast and efficient PyTables/HDF5 format.
* Memory-efficient “sparse” versions of the standard data structures for storing data that is mostly missing or mostly constant (some fixed value)
* Moving window statistics (rolling mean, rolling standard deviation, etc.)
* Static and moving window linear and [panel regression](http://en.wikipedia.org/wiki/Panel_data)

All pandas data structures are **value-mutable (the values they contain can be altered)** but not always size-mutable.

**DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object. Like Series, DataFrame accepts many different kinds of input:

* Dict of 1D ndarrays, lists, dicts, or Series
* 2-D numpy.ndarray
* [Structured or record](http://docs.scipy.org/doc/numpy/user/basics.rec.html) ndarray ( )
* A Series (Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called *index*. Pandas Series is nothing but a column in an excel sheet.
* Another DataFrame

**Read a comma-separated values (csv) file into DataFrame.**

**Pandas.read\_csv**(*filepath\_or\_buffer*, *sep=<no\_default>*, *delimiter=None*, *header='infer'*, *names=<no\_default>*, *index\_col=None*, *usecols=None*, *squeeze=False*, *prefix=<no\_default>*, *mangle\_dupe\_cols=True*, *dtype=None*, *engine=None*, *converters=None*, *true\_values=None*, *false\_values=None*, *skipinitialspace=False*, *skiprows=None*, *skipfooter=0*, *nrows=None*, *na\_values=None*, *keep\_default\_na=True*, *na\_filter=True*, *verbose=False*, *skip\_blank\_lines=True*, *parse\_dates=False*, *infer\_datetime\_format=False*, *keep\_date\_col=False*, *date\_parser=None*, *dayfirst=False*, *cache\_dates=True*, *iterator=False*, *chunksize=None*, *compression='infer'*, *thousands=None*, *decimal='.'*, *lineterminator=None*, *quotechar='"'*, *quoting=0*, *doublequote=True*, *escapechar=None*, *comment=None*, *encoding=None*, *encoding\_errors='strict'*, *dialect=None*, *error\_bad\_lines=None*, *warn\_bad\_lines=None*, *on\_bad\_lines=None*, *delim\_whitespace=False*, *low\_memory=True*, *memory\_map=False*, *float\_precision=None*, *storage\_options=None*)

Few important parameters are discussed below:

**filepath\_or\_buffer: *str, path object or file-like object***

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, gs, and file. For file URLs, a host is expected. A local file could be: file://localhost/path/to/table.csv.

**Sep: *str, default ‘,’***

Delimiter to use. If sep is None, the C engine cannot automatically detect the separator, but the Python parsing engine can, meaning the latter will be used and automatically detect the separator by Python’s builtin sniffer tool, csv.Sniffer. In addition, separators longer than 1 character and different from '\s+' will be interpreted as regular expressions and will also force the use of the Python parsing engine. Note that regex delimiters are prone to ignoring quoted data. Regex example: '\r\t'.

**Header: *int, list of int, default ‘infer’***

Row number(s) to use as the column names, and the start of the data. Default behavior is to infer the column names: if no names are passed the behavior is identical to header=0 and column names are inferred from the first line of the file, if column names are passed explicitly then the behavior is identical to header=None. Explicitly pass header=0 to be able to replace existing names. The header can be a list of integers that specify row locations for a multi-index on the columns e.g. [0,1,3]. Intervening rows that are not specified will be skipped (e.g. 2 in this example is skipped). Note that this parameter ignores commented lines and empty lines if skip\_blank\_lines=True, so header=0 denotes the first line of data rather than the first line of the file.

**index\_col*int, str, sequence of int / str, or False, default None***

Column(s) to use as the row labels of the DataFrame, either given as string name or column index. If a sequence of int / str is given, a MultiIndex is used.

Note: index\_col=False can be used to force pandas to *not* use the first column as the index, e.g. when you have a malformed file with delimiters at the end of each line.

**Read an Excel file into a pandas DataFrame.**

**pandas.read\_excel**(*io*, *sheet\_name=0*, *header=0*, *names=None*, *index\_col=None*, *usecols=None*, *squeeze=False*, *dtype=None*, *engine=None*, *converters=None*, *true\_values=None*, *false\_values=None*, *skiprows=None*, *nrows=None*, *na\_values=None*, *keep\_default\_na=True*, *na\_filter=True*, *verbose=False*, *parse\_dates=False*, *date\_parser=None*, *thousands=None*, *comment=None*, *skipfooter=0*, *convert\_float=None*, *mangle\_dupe\_cols=True*, *storage\_options=None*)

**Head and Tail:**

To view a small sample of a Series or DataFrame object, use the head and tail methods. The default number of elements to display is five, but you may pass a custom number.

Example:

df = pd.DataFrame({'animal': ['alligator', 'bee', 'falcon', 'lion',

**...**  'monkey', 'parrot', 'shark', 'whale', 'zebra']})

**>>>** df

Output:

animal

0 alligator

1 bee

2 falcon

3 lion

4 monkey

5 parrot

6 shark

7 whale

8 zebra

>>>df.head() # check the output

>>>df.head(3) # check the output

animal

0 alligator

1 bee

2 falcon

**df.columns** is used to get column labels. Try this command on your dataset. Also try following commands: **df.shape, df.size, df.ndim. df.memory\_usage().**

**DataFrame.isna()**

Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or **numpy.NaN**, gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings '' or **numpy.inf** are not considered NA values

df = pd.DataFrame(dict (age=[5, 6, np.NaN],

**...**  born=[pd.NaT, pd.Timestamp('1939-05-27'),

**...**  pd.Timestamp('1940-04-25')],

**...**  name=['Alfred', 'Batman', ''],

**...**  toy=[**None**, 'Batmobile', 'Joker']))

**>>>** df

age born name toy

0 5.0 NaT Alfred None

1 6.0 1939-05-27 Batman Batmobile

2 NaN 1940-04-25 Joker

>>>df.isna()

age born name toy

0 False True False True

1 False False False False

2 True False False False

**DataFrame.dropna(*axis=0*, *how='any'*, *thresh=None*, *subset=None*, *inplace=False*)**

Dropna is used to remove missing values.

See the [User Guide](https://pandas.pydata.org/docs/user_guide/missing_data.html" \l "missing-data) for more on which values are considered missing, and how to work with missing data.

**Parameters**

**axis*{0 or ‘index’, 1 or ‘columns’}, default 0***

Determine if rows or columns which contain missing values are removed.

* 0, or ‘index’ : Drop rows which contain missing values.
* 1, or ‘columns’ : Drop columns which contain missing value.

**how*{‘any’, ‘all’}, default ‘any’***

Determine if row or column is removed from DataFrame, when we have at least one NA or all NA.

* ‘any’ : If any NA values are present, drop that row or column.
* ‘all’ : If all values are NA, drop that row or column.

**thresh*int, optional***

Require that many non-NA values.

**subset*array-like, optional***

Labels along other axis to consider, e.g. if you are dropping rows these would be a list of columns to include.

**inplace*bool, default False***

If True, do operation inplace and return None.

df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],

**...**  "toy": [np.nan, 'Batmobile', 'Bullwhip'],

**...**  "born": [pd.NaT, pd.Timestamp("1940-04-25"),

**...**  pd.NaT]})

df

name toy born

0 Alfred NaN NaT

1 Batman Batmobile 1940-04-25

2 Catwoman Bullwhip NaT

df.dropna()

name toy born

1 Batman Batmobile 1940-04-25

# Drop the columns where at least one element is missing.

df.dropna(axis='columns')

name

0 Alfred

1 Batman

2 Catwoman

Drop the rows where all elements are missing.

df.dropna(how='all')

name toy born

0 Alfred NaN NaT

1 Batman Batmobile 1940-04-25

2 Catwoman Bullwhip NaT

**pandas.dataframe.loc**

**.loc[]** is primarily label based, **The loc operator is used to index a portion of the dataframe**. Allowed inputs are:

* A single label, e.g. 5 or 'a', (note that 5 is interpreted as a label of the index, and **never** as an integer position along the index).
* A list or array of labels, e.g. ['a', 'b', 'c'].
* A slice object with labels, e.g. 'a':'f'.

df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],

**...**  index=['cobra', 'viper', 'sidewinder'],

**...**  columns=['max\_speed', 'shield'])

df

max\_speed shield

cobra 1 2

viper 4 5

sidewinder 7 8

df.loc['viper']

max\_speed 4

shield 5

Name: viper, dtype: int64

df.loc[['viper', 'sidewinder']]

max\_speed shield

viper 4 5

sidewinder 7 8

Slice with labels for row and single label for column. As mentioned above, note that both the start and stop of the slice are included.

df.loc['cobra':'viper', 'max\_speed']

cobra 1

viper 4

Name: max\_speed, dtype: int64

**Pandas.dataframe.iloc :** Purely integer-location based indexing for selection by position.

.iloc[] is primarily integer position based (from 0 to length-1 of the axis), but may also be used with a boolean array.

Allowed inputs are:

* An integer, e.g. 5.
* A list or array of integers, e.g. [4, 3, 0].
* A slice object with ints, e.g. 1:7.
* A boolean array.
* A callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above). This is useful in method chains, when you don’t have a reference to the calling object, but would like to base your selection on some value.
* mydict = [{'a': 1, 'b': 2, 'c': 3, 'd': 4},
* **...**  {'a': 100, 'b': 200, 'c': 300, 'd': 400},
* **...**  {'a': 1000, 'b': 2000, 'c': 3000, 'd': 4000 }]
* **>>>** df = pd.DataFrame(mydict)
* **>>>** df
* a b c d
* 0 1 2 3 4
* 1 100 200 300 400
* 2 1000 2000 3000 4000
* df.iloc[[0]]
* a b c d
* 0 1 2 3 4
* **>>>** type(df.iloc[[0]])
* <class 'pandas.core.frame.DataFrame'>
* df.iloc[[0, 1]]
* a b c d
* 0 1 2 3 4
* 1 100 200 300 400

Try following commands on your dataset. Here ‘df’ indicates dataframe

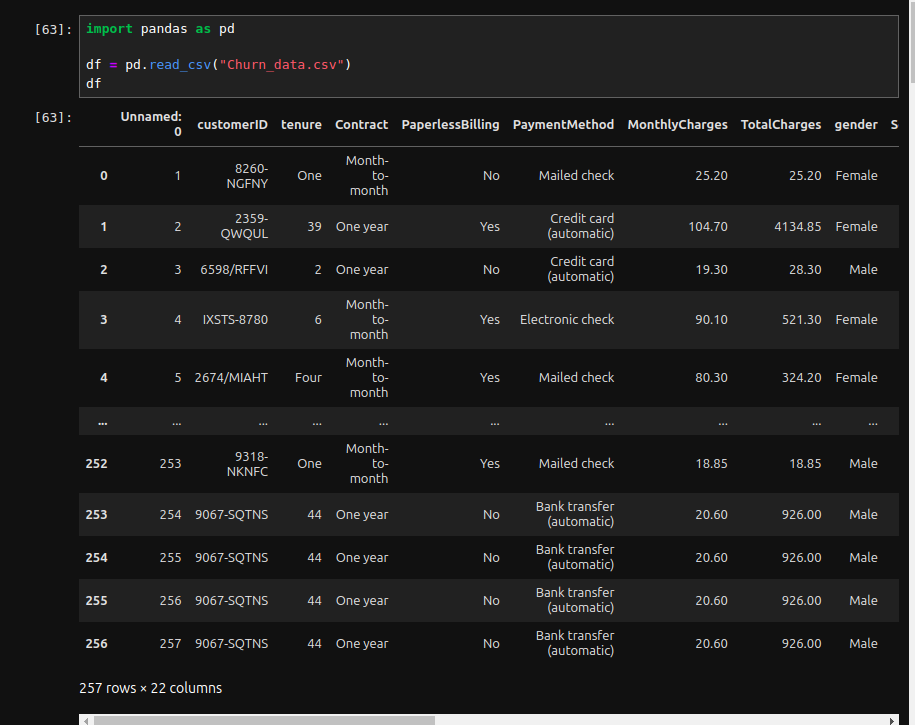
**df['column\_name'].unique()** gives unique values in Species column

**data['column\_name'].value\_counts()** returns a count of unique rows in dataframe

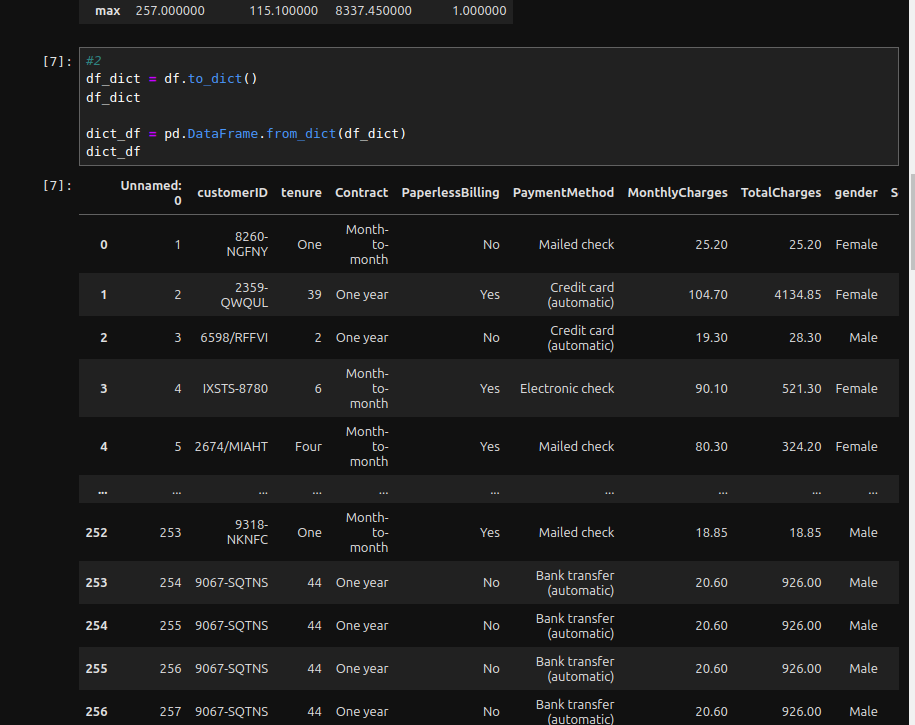
**df.isnull().sum()** returns total null values in each column.

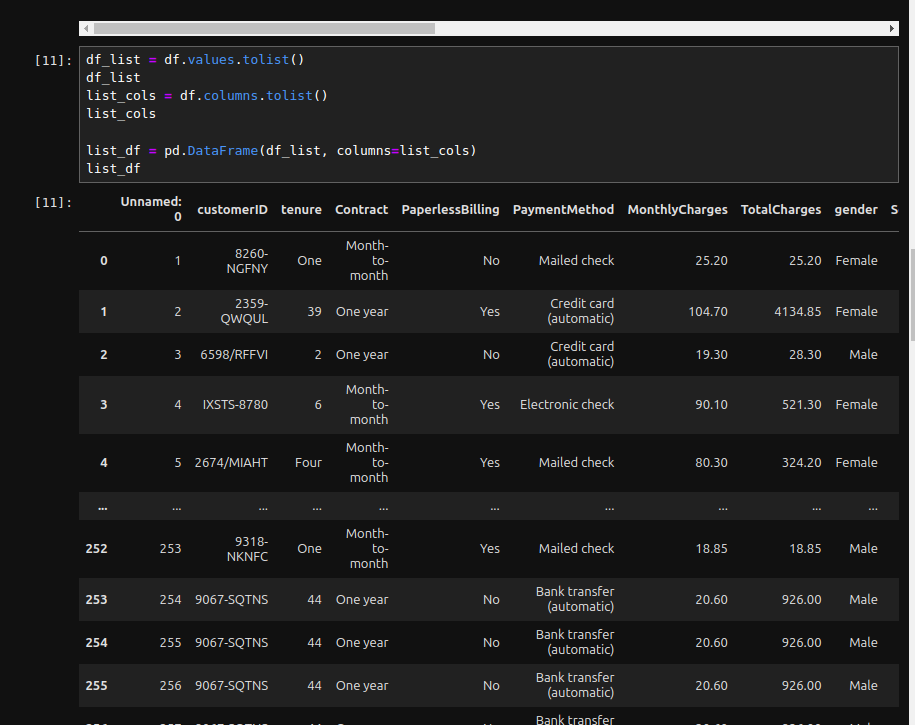
**A.5 Task:**

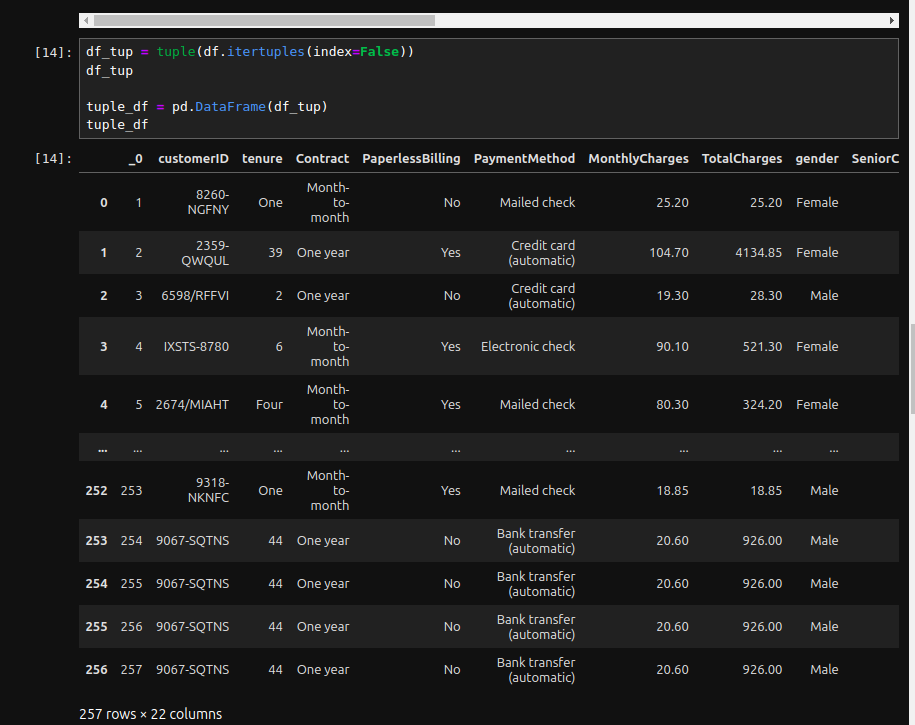
1. Select dataset with (.csv file) of sufficiently large size (>300 rows). It should contain columns with different data types.(churnt data set)



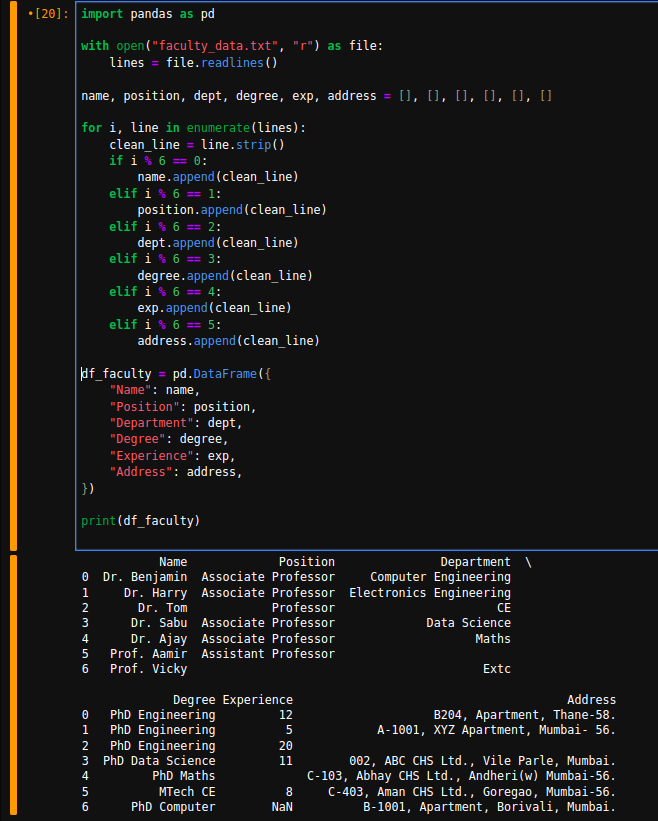
1. Create data frames from data structures (dictionary, list, tuple)



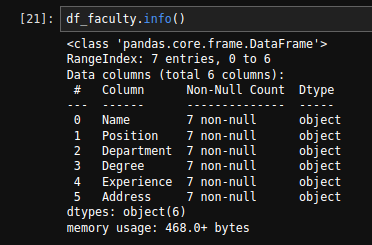




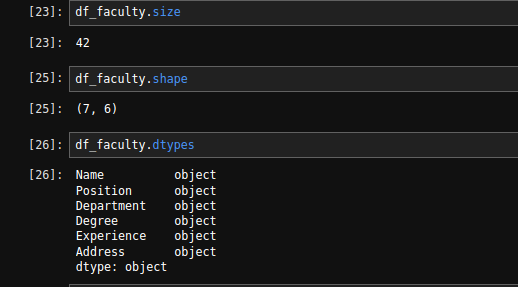
1. Create data frames from unformatted text file.



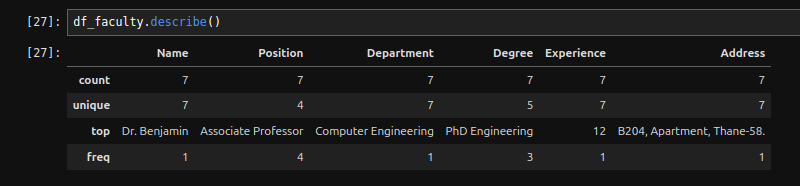
1. Obtain metadata of given dataset



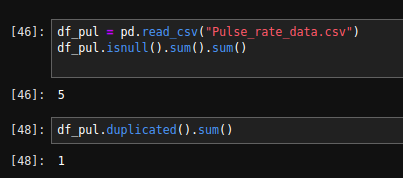
1. Explore size, shape, data types of each column in the dataset.



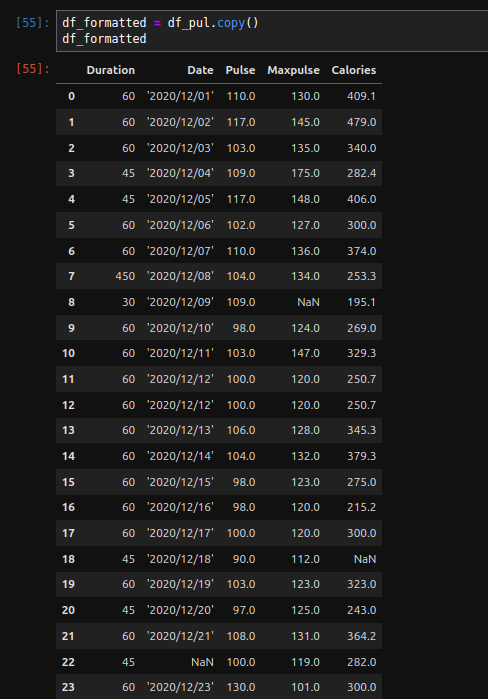
1. Perform statistical analysis with min, max, average, standard deviation, percentile etc.

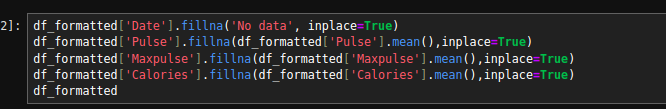


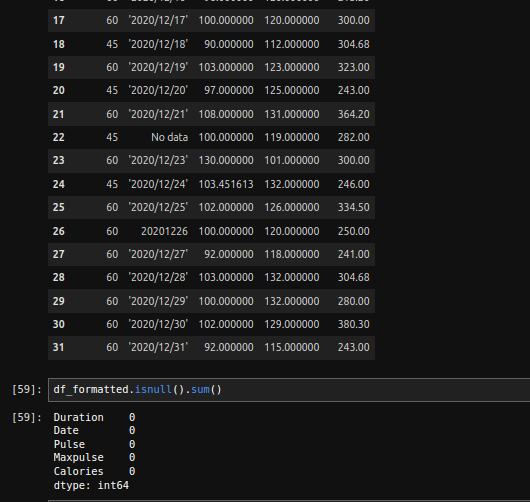
1. Identify null values, missing values in dataset(pulse rate data)



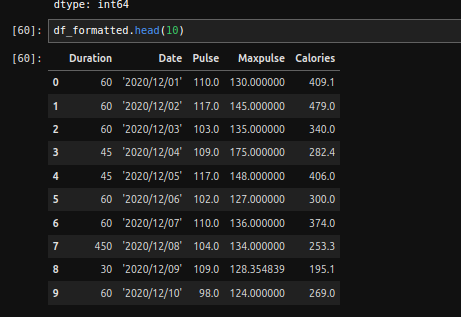
1. Replace missing values by appropriate values.
2. Remove null values from the dataset.



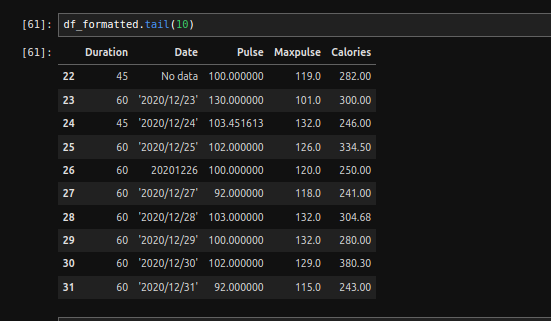


­

1. Display first 10 rows of dataset.



1. Display last 5 rows of dataset.



PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No: **A046** | Name: **TEJAS PADMAKAR** |
| Class: MCA | Batch: **2** |
| Date of Experiment: **16-02-2023** | Date of Submission: **16-02-2023** |
| Grade: |  |

**B.3 Observations and learning:**

**B.4 Conclusion:**

In this experiment, I performed Exploratory Data Analysis on real world dataset using Pandas library.

**B.5 Question of Curiosity**

1. Explore the commands which are not described in the document and you have used for exploring your selected dataset. List those commands here and describe it in brief.

Reference Links:

Reading file - <https://colab.research.google.com/drive/1_azqOSxG2K9raRV2-CQ22p9oLAB2-P28?usp=sharing>

Reading and Converting text file –

<https://colab.research.google.com/drive/12XQXDK_3ha4j544oHii712yQdqdpue92?usp=sharing>

handling missing values:

<https://colab.research.google.com/drive/1bwaBj1Rm_8V6Vx77fFm20mGOuuVjKLSb?usp=sharing>

Manipulating dataframe:

<https://colab.research.google.com/drive/14S_RSP35mxUx0GqQx5syoCNNk2ClP_Fl?usp=sharing>

Handling mismatched values:

<https://colab.research.google.com/drive/1Fg3V2W_zkCYskqwDzxKXUeb-B0xURcAU?usp=sharing>