Experiment No. 4

Title: Exploratory data analysis using PANDAS

Batch:A2 **Roll No:16010421063** **Experiment No.:4** **Aim:** To perform exploratory data analysis using python Pandas

---

Resources needed: Python IDE

Theory:

[illegible]rary that provides extensive means for data analysis. Data scientists often [illegible] in table formats like .csv, .tsv, or .xlsx. Pandas makes it very convenient to [illegible]yze such tabular data using SQL-like queries. Python has long been great [illegible] preparation, but less so for data analysis and modeling. *pandas* helps fill [illegible] to carry out your entire data analysis workflow in Python. In conjunction [illegible]eaborn, Pandas provides a wide range of opportunities for visual analysis [illegible]

[illegible]y:

[illegible]

Or

pip install pandas

The main data structures in Pandas are implemented with Series and DataFrame classes. The former is a one-dimensional indexed array of some fixed data type. The latter is a two-dimensional data structure - a table - where each column contains data of the same type. You can see it as a dictionary of Series instances. DataFrames are great for representing real data: rows correspond to instances (examples, observations, etc.), and columns correspond to features of these instances.

A series can be created using list ,dictionary etc. with index(implicit indexing) or without index(explicit indexing).

```
import pandas as pd
data1=pd.Series({2:'a', 1:'b', 3:'c'}) #implicit indexing
data2=pd.Series({2:'a', 1:'b', 3:'c'}, index=[1,2,3]) # explicit indexing
        #loc attribute allows indexing and slicing that always references the explicit index:
  data2.loc[2]
        #iloc attribute allows indexing and slicing that always references the implicit #Python-
```

style index
data.iloc[1]

Following are the various series related operations

- Append(): s3.append(s1) # Stitch s1 to s3
- Drop: s4.drop('e') #Delete the value whose index is e
- Addition: s4.add(s3)#addition according to the index, and it would be filled with NaN (null value) if the indexes are different.
- Subtraction: s4.sub(s3) #substraction according to the index, and it would be filled with NaN (null value) if the indexes are different.
- Multiplication: s4.mul(s3) #multiplication according to the index, and it would be filled with NaN (null value) if the indexes are different.
- Division: s4.div(s3)
- Median: s4.median()
- Sum: s4.sum()

inimum : s4.max() s4.min()

keeps track of both data (numerical as well as text), and column and row
ultiple columns of data.
ay to a pandas data frame with pd.Data frame().
s np

Frame(h)
ne:', df_h)
read data from dictionary and files as well.

Reading and writing data from files:
CSVs don't have indexes like our DataFrames, so all we need to do is just designate the index_col when reading.

```
import pandas as pd
df=pd.read_csv("C:/Users/Admin/Desktop/ADVANCED PYTHON/DATA/SalesJan2009.csv",index_col=0)
#Reading the dataset in a dataframe using Pandas
print(df)
```

To write data to a new csv file use to_csv()

```
df3.to_csv('animal.csv')
df3.to_excel('animal.xlsx', sheet_name='Sheet1')
```

Following functions of dataframe can be used to explore dataset to get summary of it.

- **info()** provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.
  df.info()
- **describe()** is used to get a summary of numeric values in your dataset. It calculates the mean, standard deviation, minimum value, maximum value, 1st percentile, 2nd percentile, 3rd percentile of the columns with numeric values. It also counts the number of variables in the dataset.
  df.describe()
  describe() can also be used on a categorical variable to get the count of rows, unique count of categories, top category, and freq of top category
  temp_df['product'].describe()
- **head()** outputs the first five rows of your DataFrame by default, but we could also pass a number as well
  print(df.head)

st five rows by default.

just a tuple of (rows, columns):

lection:

n of dataframe is a series . following functions can be used for row

selection

using square brackets will return a *Series.*

_df['product']

#selecting multiple columns

subset=temp_df[['product','price']]

accessing rows:

.loc - locates by name

prom = movies_df.loc["Prometheus"]

prom

.iloc- locates by numerical index

prod = df.iloc[1]

prod

Further analysis using pandas dataframe:

value_counts() can tell us the frequency of all values in a column.

temp_df['product'].value_counts().head(10)

nunique() to count number of unique values that occur in dataset or in a column

df.nunique() #to see the counts of unique numbers in each column df["Embarked"].nunique() #to get the unique count of a column

corr() generate the relationship between each continuous variable:

temp_df.corr()

Correlation tables are a numerical representation of the bivariate relationships in the dataset.

astype() can be used to change the datatype of that column

df["Embarked"] = df["Embarked"].astype("category")

df["Embarked"].dtype

column clean up funtions:

append() will return a copy after appending without affecting the original DataFrame(if inplace attribute is used).

temp_df = df.append(df)

temp_df.shape

nethod will return a copy of DataFramewith duplicates removed.

)_df.drop_duplicates()

id reasignment it can be done

luplicates(inplace=True)

olumn names of dataset also can be used for renaming

o delete columns

s=['A', 'C'])

used to rename certain or all columns via a dict.

```
temp_df.rename(columns={
'Account_Created': 'Acc_Created',
'Last_Login': 'Lst_Login'
 }, inplace=True)
temp_df.columns
```

Handling null values using pandas:

Mostly Python's None or NumPy's np.nan indicates missing or null values.

- **isnull()** checks which cells in our DataFrame are null. It returns a DataFrame where each cell is either True or False depending on that cell's null status.

  temp_df.isnull()

To count the number of nulls in each column we use an aggregate function .sum() for summing:

temp_df.isnull().sum()

To get rid of rows or columns with nulls. Removing null data is only suggested if you have a small amount of missing data. .dropna() will delete any row with at least a single null value, but it will return a new DataFrame without altering the original one.

    temp_df.dropna()

- Or drop columns with null values by setting axis=1.

    temp_df.dropna(axis=1)

- Replace nulls with non-null values, a technique known as imputation. Normally null value is replaced with mean or the median of that column.

Conditional selections/ Filtering

Comparison operators are used for filtering

Take a column from the DataFrame and apply a Boolean condition to it.

    condition = (movies_df['Director'] == "Ridley Scott")

It returns a Series of True and False values.Some more examples on conditionals

Select movies_df where movies_df Director equals Ridley Scott.

ies_df['Director'] == "Ridley Scott"]

ies_df['Rating'] >= 8.6].head(3)

vies_df['Director'] == 'Christopher Nolan') | (movies_df['Director'] ==

head()

ies_df['Director'].isin(['Christopher Nolan', 'Ridley Scott'])].head()

Functions/ Aggregate Functions

ns are the ones that reduce the dimension of the returned objects.

ome aggregate functions to understand the overall properties of a dataset

number of rows /items

df.mean(): To find mean average of data frame

Synatx: data.Population.mean()#where Population is column name

df.median(): To find median of data frame

df.quantile():

df.sum(): Do a summation operation on any column in the DataFrame

df.prod(): To find Product of all items

df.std():To find standard deviation of a data frame

df.var():To find varianceof data frame

df.min(), df.max(): To find Minimum and maximum

df.first(), df.last(): First and last item

GROUP BY and aggregation

―group by‖ process can involve one or more of the following steps:

–Splitting the data into groups based on some criteria.

–Applying a function to each group independently.

–Combining the results into a data structure

Apply step involves following

–Aggregation: compute a summary statistic (or statistics) for each group. Some examples:
- •Compute group sums or means.
- •Compute group sizes / counts.

–Transformation: perform some group-specific computations and return a like-indexed object. Some examples:
- •Standardize data (zscore) within a group.
- •Filling NAs within groups with a value derived from each group.

–Filtration: discard some groups, according to a group-wise computation that evaluates True or False. Some examples:
- •Discard data that belongs to groups with only a few members.
- •Filter out data based on the group sum or mean.

```
# group the data on team value.
    gk = df.groupby('Team')
```

contained in the "Boston Celtics" group

oston Celtics')

rame or Series we can use list, but doing so — especially on large datasets

is to apply() a function to the dataset. Using apply() will be much faster

over rows because pandas is utilizing vectorization.

Combining Datasets

Concat: s to append either columns or rows from one DataFrame to another.

Joining two dataframe on the index

merge two dataframes on key attribute

Activities:

1. Download data set with atleast 1500 rows and 10-20 columns(numeric and non numeric) from valid data sources
2. Read same in pandas DataFrame
3. Perform in detail Exploratory data analysis of this dataset
   - Get information and description of dataset.

- See if any null values are present. Display count of null values.
- Choose the appropriate technique to handle missing values.(imputation with use of inplace)
- Use sorting of data in dataframe to display topmost 5 or 8 records based on one or more column values(conditional filtering)
- Get frequency listing of any one relavant column(2 cases)
- Sorting of rows and columns,(implicit and explicit indexing)
- Accessing particular row based on certain condition and displaying only one or few columns from it.(3 cases with compound conditions)
- Minimum and maximum values related analysis
- Use of group by on one or more columns(2 cases)
- Add new column to existing dataframe and populate same using existing columns data.
- Use of appropriate aggregate functions with groupby.(2 cases)
- Selection on particular groups based on name or condition
- Find correlation between any two columns values.
- Try transformation(normalization using any technique) on data set
- Joining , merging and concatenation of data in dataframe.

Write down observation for your dataset for each of above listed task of analysis.

Result: (script and output)

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

[2] ✓ 0.4s

```
df=pd.read_csv("data.csv")
```

[3] ✓ 0.3s

```
df.describe()
```

[4] ✓ 0.9s

	followers	Likes	Boost Index	Engagement Rate	Engagement Rate 60days	V
count	8.570000e+02	8.570000e+02	857.000000	855.000000	857.000000	8.570000
mean	4.952975e+07	2.330995e+08	67.394399	0.352958	0.074128	2.70591
std	2.861902e+07	3.862461e+08	14.963222	0.798509	0.176490	2.79723
min	2.400000e+07	0.000000e+00	1.000000	0.000261	0.000000	0.000000
25%	3.310000e+07	2.926386e+07	62.000000	0.026044	0.005435	1.33829
50%	4.130000e+07	9.537514e+07	71.000000	0.098933	0.020671	2.01949
75%	5.340000e+07	2.351904e+08	78.000000	0.379554	0.062921	2.83470
max	2.200000e+08	2.191406e+09	88.000000	10.584084	1.519044	1.95660

```
df.isnull().sum()
```

[5] ✓ 0.5s

```
Country                150
Channel Name             0
Category               121
Main Video Category      2
username                 0
followers                0
Main topic               2
More topics              2
Likes                    0
Boost Index              0
Engagement Rate          2
```

```
df1=df[['Channel Name','followers','Likes','Boost Index','Engagement Ra
Eng_mean=df1['Engagement Rate'].mean()
df1['Engagement Rate'].fillna(value=Eng_mean,inplace=True)
df1.isnull().sum()
```

[6] ✓ 0.2s

```
/tmp/ipykernel_10978/2046750724.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs
  df1['Engagement Rate'].fillna(value=Eng_mean,inplace=True)

Channel Name       0
followers          0
Likes              0
Boost Index        0
Engagement Rate    0
dtype: int64
```

```
df1[df1['Boost Index']>50].sort_values(by=['Boost Index']).head(5)
```

[7] ✓ 0.4s

	Channel Name	followers	Likes	Boost Index	Engagement Rate
71	Voot Kids	38100000	46360335.99	51	0.098878
721	Voot Kids	38100000	46360335.99	51	0.098878
521	Voot Kids	38100000	46360335.99	51	0.098878
221	Voot Kids	38100000	46360335.99	51	0.098878
296	Voot Kids	38100000	46360335.99	51	0.098878

```
df['Category'].value_counts()
```

[8] ✓ 0.2s

```
Gaming & Apps       398
Music               210
None                 62
Beauty & Fashion     36
```

```
[9]  df['Main topic'].value_counts()
     ✓ 0.2s

...  Music                          168
     Entertainment                  138
     Lifestyle                      135
     Movies                          96
     Pop music                       64
     Music of Asia                   47
     Hip hop music                   34
     TV shows                        28
     Action game                     22
     Hobby                           21
     Technology                      18
     Gaming                          18
     Society                         14
     Electronic music                14
     Music of Latin America          14
     Rhythm and blues                 6
     Action-adventure game            4
     Role-playing video game          4
     Food                             2
     Knowledge                        2
     Strategy video game              2
     Vehicles                         2
     Rock music                       2
     Name: Main topic, dtype: int64
```

```
[14] df[(df['Main topic']=='Entertainment') | (df['Main topic']=='Movies')][
     ✓ 0.2s
```

	Channel Name	Main topic
1	ABCkidTV - Nursery Rhymes	Movies
2	SET India	Movies
8	ABCkidTV - Nursery Rhymes	Movies
9	SET India	Movies
19	Goldmines Telefilms	Movies

```
df[(df['Main topic']=='Entertainment') | (df['Main topic']=='Movies')][["Channel Name", 'Main topic']]
```

[14] ✓ 0.2s

	Channel Name	Main topic
1	ABCkidTV - Nursery Rhymes	Movies
2	SET India	Movies
8	ABCkidTV - Nursery Rhymes	Movies
9	SET India	Movies
19	Goldmines Telefilms	Movies
...	...	...
836	Smosh	Entertainment
840	Nick Jr.	Movies
847	WatchMojo.com	Entertainment
848	Super JoJo - Nursery Rhymes & Kids Songs	Movies
854	Amit Bhadana	Entertainment

234 rows × 2 columns

```
df['Engagement Rate'].max()
```

[16] ✓ 0.2s

```
10.58408389
```

```
df["Engagement Rate"].min()
```

[17] ✓ 0.2s

```
0.0002609916678
```

```
df[["Main topic","Views"]].groupby(["Main topic"]).sum()
```

[20] ✓ 0.2s

	Views
Main topic	
Action game	192774528416

```
df[["Main topic","Views"]].groupby(["Main topic"]).sum()
```

[20] ✓ 0.2s

Main topic	Views
Action game	192774528416
Action-adventure game	32001873040
Electronic music	196298935862
Entertainment	3392077258994
Food	37673672622
Gaming	224938301538
Hip hop music	636202123048
Hobby	1300298209511
Knowledge	13111393572
Lifestyle	3264219428118
Movies	4509505323184
Music	3791766520248
Music of Asia	2570390687728
Music of Latin America	265119014468
Pop music	1515408509942
Rhythm and blues	95320111054
Rock music	37226274172
Role-playing video game	43577694002
Society	226016948574
Strategy video game	9488615174
TV shows	696702472446
Technology	130629918392
Vehicles	8961060020

```
df["Social Score"]=df['Views']*df["Engagement Rate"]
df["Social Score"]
```

[21] ✓ 0.4s

```
0      6.547421e+09
1      8.536444e+10
```

```
df["Social Score"]=df['Views']*df["Engagement Rate"]
df["Social Score"]
```

[21] ✓ 0.4s

```
0      6.547421e+09
1      8.536444e+10
2      1.468534e+08
3      1.802831e+09
4      1.184428e+10
           ...
852    3.312732e+08
853    1.873949e+10
854    2.262326e+09
855    1.488611e+09
856    1.705339e+08
Name: Social Score, Length: 857, dtype: float64
```

```
df[["Main topic","Views"]].groupby(["Main topic"]).mean()
```

[22] ✓ 0.3s

	Views
Main topic	
Action game	8.762479e+09
Action-adventure game	8.000468e+09
Electronic music	1.402135e+10
Entertainment	2.458027e+10
Food	1.883684e+10
Gaming	1.249657e+10
Hip hop music	1.871183e+10
Hobby	6.191896e+10
Knowledge	6.555697e+09
Lifestyle	2.417940e+10
Movies	4.697401e+10
Music	2.257004e+10
Music of Asia	5.468916e+10
Music of Latin America	1.893707e+10

```
df[["Main topic","Views"]].groupby(["Main topic"]).var()
```

[23] ✓ 0.7s

Main topic	Views
Action game	6.328067e+18
Action-adventure game	3.862837e+19
Electronic music	6.529738e+18
Entertainment	3.312920e+20
Food	0.000000e+00
Gaming	2.109380e+19
Hip hop music	5.450168e+19
Hobby	6.078228e+20
Knowledge	0.000000e+00
Lifestyle	4.550262e+20
Movies	1.720865e+21
Music	5.898374e+19
Music of Asia	4.836831e+21
Music of Latin America	3.874523e+19
Pop music	1.008420e+20
Rhythm and blues	7.007663e+18
Rock music	0.000000e+00
Role-playing video game	8.211860e+19
Society	1.741840e+19
Strategy video game	0.000000e+00
TV shows	2.204630e+20
Technology	2.989656e+19
Vehicles	0.000000e+00

```
viewsum=df[["Main topic","Views"]].groupby(["Main topic"]).sum()
viewsum[viewsum['Views']>224938301538]
```

[26] ✓ 0.4s

	Views

```
viewsum=df[["Main topic","Views"]].groupby(["Main topic"]).sum()
viewsum[viewsum['Views']>224938301538]
```

[26] ✓ 0.4s

Main topic	Views
Entertainment	3392077258994
Hip hop music	636202123048
Hobby	1300298209511
Lifestyle	3264219428118
Movies	4509505323184
Music	3791766520248
Music of Asia	2570390687728
Music of Latin America	265119014468
Pop music	1515408509942
Society	226016948574
TV shows	696702472446

```
df.corr()
```

[27] ✓ 0.5s

```
/tmp/ipykernel_10978/1134722465.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is
only valid columns or specify the value of numeric_only to silence this warning.
  df.corr()
```

	followers	Likes	Boost Index	Engagement Rate	Engagement Rate 60days	Views	Views Avg.	Avg. 1 Day
followers	1.000000	0.500335	0.218396	-0.082989	0.019658	0.847929	0.145527	0.252499
Likes	0.500335	1.000000	0.238953	-0.028733	0.160202	0.198697	0.089432	0.347254
Boost Index	0.218396	0.238953	1.000000	-0.063909	-0.038517	0.211831	0.013309	0.053000
Engagement Rate	-0.082989	-0.028733	-0.063909	1.000000	0.436098	-0.046493	0.837483	0.003510
Engagement Rate 60days	0.019658	0.160202	-0.038517	0.436098	1.000000	-0.049838	0.553659	0.648897
Views	0.847929	0.198697	0.211831	-0.046493	-0.049838	1.000000	0.124365	0.185815
Views Avg.	0.145527	0.089432	0.013309	0.837483	0.553659	0.124365	1.000000	0.171712
Avg. 1 Day	0.252499	0.347254	0.053000	0.003510	0.648897	0.185815	0.171712	1.000000
Avg. 3 Day	0.318546	0.390695	0.076731	0.134928	0.746745	0.258719	0.424038	0.779004

```
df.corr()
```

[27] ✓ 0.5s

```
/tmp/ipykernel_10978/1134722465.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is
only valid columns or specify the value of numeric_only to silence this warning.
  df.corr()
```

	followers	Likes	Boost Index	Engagement Rate	Engagement Rate 60days	Views	Views Avg.	Avg. 1 Day
followers	1.000000	0.500335	0.218396	-0.082989	0.019658	0.847929	0.145527	0.252499
Likes	0.500335	1.000000	0.238953	-0.028733	0.160202	0.198697	0.089432	0.347254
Boost Index	0.218396	0.238953	1.000000	-0.063909	-0.038517	0.211831	0.013309	0.053000
Engagement Rate	-0.082989	-0.028733	-0.063909	1.000000	0.436098	-0.046493	0.837483	0.003510
Engagement Rate 60days	0.019658	0.160202	-0.038517	0.436098	1.000000	-0.049838	0.553659	0.648897
Views	0.847929	0.198697	0.211831	-0.046493	-0.049838	1.000000	0.124365	0.185815
Views Avg.	0.145527	0.089432	0.013309	0.837483	0.553659	0.124365	1.000000	0.171712
Avg. 1 Day	0.252499	0.347254	0.053000	0.003510	0.648897	0.185815	0.171712	1.000000
Avg. 3 Day	0.318546	0.390695	0.076731	0.134928	0.746745	0.258719	0.424038	0.779004
Avg. 7 Day	0.305670	0.494882	-0.011208	0.171919	0.836151	0.060090	0.520776	0.798947
Avg. 14 Day	0.129723	0.230142	-0.001729	0.435554	0.941751	0.033970	0.581435	0.783053
Avg. 30 day	0.168446	0.210411	0.029677	0.371485	0.945966	0.066084	0.566698	0.823114
Avg. 60 day	0.244095	0.310718	0.002962	0.342436	0.875859	0.104502	0.620467	0.793366
Comments Avg	0.030719	0.457865	-0.051820	0.208010	0.305790	-0.221748	0.235357	0.137436
Social Score	0.207117	-0.026645	0.030513	0.806315	0.366369	0.295678	0.884210	0.357316

```
df['Avg. 1 Day']=df['Avg. 1 Day']/df['Avg. 1 Day'].max()
df['Avg. 1 Day']
```

[28] ✓ 0.2s

```
0      0.043841
1      0.529256
2           NaN
3           NaN
4           NaN
         ...
852    0.020147
853         NaN
```

Comments Avg	0.030719	0.457865	-0.051820	0.208010	0.305790	-0.221748	0.235357	0.137436
Social Score	0.207117	-0.026645	0.030513	0.806315	0.366369	0.295678	0.884210	0.357316

```
df['Avg. 1 Day']=df['Avg. 1 Day']/df['Avg. 1 Day'].max()
df['Avg. 1 Day']
```

[28] 0.2s

```
0      0.043841
1      0.529256
2           NaN
3           NaN
4           NaN
         ...
852    0.020147
853         NaN
854    0.000000
855         NaN
856    0.006274
Name: Avg. 1 Day, Length: 857, dtype: float64
```

[ ]

Outcomes: Inculcate the knowledge of python libraries like numpy,pandas,matplotlib for scientific- computing and data visualization.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

We understood the concepts of python pandas and performed operations on dataset.

References:

ckle, Learning Python Testing, Packt Publishing, 1st Edition, 2014
Core Python Applications Programming, O’Reilly, 3rd Edition, 2015
, Python for Data Analysis, O’Reilly, 1st Edition, 2017
wsk, MySQL for Python, Packt Publishing, 1st Edition, 2010
stering Python Networking, Packt Publishing, 2nd Edition, 2017