



Experiment No. 2

Title: Exploratory data analysis using PANDAS



Batch:B1

Roll No:16010420133

Experiment No.:2

Aim: To perform exploratory data analysis using python Pandas

Resources needed: Python IDE

Theory:

Pandas is a Python library that provides extensive means for data analysis. Data scientists often work with data stored in table formats like .csv, .tsv, or .xlsx. Pandas makes it very convenient to load, process, and analyze such tabular data using SQL-like queries. Python has long been great for data munging and preparation, but less so for data analysis and modeling. *pandas* helps fill this gap, enabling you to carry out your entire data analysis workflow in Python. In conjunction

with Matplotlib and Seaborn, Pandas provides a wide range of opportunities for visual analysis of tabular data.

Installing pandas library:

Conda install pandas

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 twodimensional 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
#Pythonstyle 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)` #subtraction 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()`
- `Maximum & Minimum : s4.max() s4.min()`

A data frame object keeps track of both data (numerical as well as text), and column and row headers. It can have multiple columns of data.

convert a numpy array to a pandas data frame with `pd.DataFrame()`.

```
import numpy as np
h = [[1,2],[3,4]]
df_h = pd.DataFrame(h)
print('Data Frame:', df_h)
```

data frame can 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)`
- **tail()** outputs last five rows by default. `print(df.tail)`
- **shape()** outputs just a tuple of (rows, columns): `df.shape`

Row and column selection:

Each row and column of dataframe is a series . following functions can be used for row selection and column selection

Extracting a column using square brackets will return a *Series*.

```
prod_col=temp_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`

drop_duplicates() method will return a copy of DataFrame with duplicates removed.

```
temp_df = temp_df.drop_duplicates()
#inorder to avoid reassignment it can be done
temp_df.drop_duplicates(inplace=True)
temp_df.shape
```

.columns print the column names of dataset also can be used for renaming

```
temp_df.columns
```

drop() can be used to delete columns

```
df.drop(columns=['A', 'C'])
```

rename() method is 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.

(Autonomous College Affiliated to University of Mumbai)

```
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.

```
movies_df[movies_df['Director'] == "Ridley Scott"]
movies_df[movies_df['Rating'] >= 8.6].head(3)
movies_df[(movies_df['Director'] == 'Christopher Nolan') | (movies_df['Director'] ==
'Ridley Scott')].head()
movies_df[movies_df['Director'].isin(['Christopher Nolan', 'Ridley Scott'])].head()
```

Summary statistics Functions/ Aggregate Functions

Aggregating functions are the ones that reduce the dimension of the returned objects.

DataFrames include some aggregate functions to understand the overall properties of a dataset

df.count(): Count the number of rows /items

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

Syntax: 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')
# Finding the values contained in the "Boston Celtics" group
gk.get_group('Boston Celtics')
```

Applying functions

To iterate over a DataFrame or Series we can use list, but doing so — especially on large datasets — is very slow.

An efficient alternative is to apply() a function to the dataset. Using apply() will be much faster than iterating manually over rows because pandas is utilizing vectorization. Combining datasets:

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 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 relevant 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)
 - Adding new column to existing dataframe and populating 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.

(Autonomous College Affiliated to University of Mumbai)

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

Result: (script and output)

```
# Reading the csv data into a dataframe
data = pd.read_csv("googleplaystore.csv")
data.head(10)
```

+ Code + Text Connect Editing

```
[ ] # Reading the csv data into a dataframe
data = pd.read_csv("googleplaystore.csv")
data.head(10)
```

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres | Last Updated | Current Ver | Android Ver |
|---|---|----------------|--------|---------|------|-------------|------|-------|----------------|---------------------------|--------------------|--------------------|--------------|
| 0 | Photo Editor & Candy Camera & Grid & ScrapBook | ART_AND_DESIGN | 4.1 | 159 | 19M | 10,000+ | Free | 0 | Everyone | Art & Design | January 7, 2018 | 1.0.0 | 4.0.3 and up |
| 1 | Coloring book moana | ART_AND_DESIGN | 3.9 | 967 | 14M | 500,000+ | Free | 0 | Everyone | Art & Design;Pretend Play | January 15, 2018 | 2.0.0 | 4.0.3 and up |
| 2 | U Launcher Lite – FREE Live Cool Themes, Hide ... | ART_AND_DESIGN | 4.7 | 87510 | 8.7M | 5,000,000+ | Free | 0 | Everyone | Art & Design | August 1, 2018 | 1.2.4 | 4.0.3 and up |
| 3 | Sketch - Draw & Paint | ART_AND_DESIGN | 4.5 | 215644 | 25M | 50,000,000+ | Free | 0 | Teen | Art & Design | June 8, 2018 | Varies with device | 4.2 and up |
| 4 | Pixel Draw - Number Art Coloring Book | ART_AND_DESIGN | 4.3 | 967 | 2.8M | 100,000+ | Free | 0 | Everyone | Art & Design;Creativity | June 20, 2018 | 1.1 | 4.4 and up |
| 5 | Paper flowers instructions | ART_AND_DESIGN | 4.4 | 167 | 5.6M | 50,000+ | Free | 0 | Everyone | Art & Design | March 26, 2017 | 1.0 | 2.3 and up |
| 6 | Smoke Effect Photo Maker - Smoke Editor | ART_AND_DESIGN | 3.8 | 178 | 19M | 50,000+ | Free | 0 | Everyone | Art & Design | April 26, 2018 | 1.1 | 4.0.3 and up |
| 7 | Infinite Painter | ART_AND_DESIGN | 4.1 | 36815 | 29M | 1,000,000+ | Free | 0 | Everyone | Art & Design | June 14, 2018 | 6.1.61.1 | 4.2 and up |
| 8 | Garden Coloring Book | ART_AND_DESIGN | 4.4 | 13791 | 33M | 1,000,000+ | Free | 0 | Everyone | Art & Design | September 20, 2017 | 2.9.2 | 3.0 and up |
| 9 | Kids Paint Free - Drawing Fun | ART_AND_DESIGN | 4.7 | 121 | 3.1M | 10,000+ | Free | 0 | Everyone | Art & Design;Creativity | July 3, 2018 | 2.8 | 4.0.3 and up |

```
# Checking NaN values
data.isnull().sum()

# Cleaning of 'Rating' data
t_data = data[pd.notnull(data['Rating'])]

# Finding the mean
mean = np.mean(t_data['Rating'])

print("Mean of ratings excluding null values:", mean)
```


Data Cleaning

```
[ ] # Checking NaN values
data.isnull().sum()
```

```
App          0
Category     0
Rating      1474
Reviews      0
Size         0
Installs     0
Type         1
Price        0
Content Rating 1
Genres       0
Last Updated 0
Current Ver  8
Android Ver  3
dtype: int64
```

```
# Cleaning of 'Rating' data
t_data = data[pd.notnull(data['Rating'])]
```

```
# Finding the mean
mean = np.mean(t_data['Rating'])
```

```
print("Mean of ratings excluding null values:", mean)
```

```
Mean of ratings excluding null values: 4.193338315362448
```

```
# There are many missing values in 'Rating' column so replace NaN with mean
data['Rating'].fillna(round(mean, 1), inplace = True)
```

```
# In other columns missing value is not considerable so drop NaNs
data.dropna(inplace = True)
```

```
# Finding the minimum, maximum, average, median, range, mode, count and variance
of the number of installs of all apps
```

```
df = data
maximum = np.max(df['Installs'])
minimum = np.min(df['Installs'])
avg = np.mean(df['Installs'])
med = np.median(df['Installs'])
r = maximum - minimum
var = np.var(df['Installs'])
```

```
print("Maximum number of installs of an App on Play Store: " + str(maximum))
print("Minimum number of installs of an App on Play Store: " + str(minimum))
print("Average number of installs of an App on Play Store: " + str(round(avg, 2))
)
print("Median of number of installs of Apps on Play Store: " + str(round(med, 2))
)
print("Range of number of installs of Apps on Play Store: " + str(r))
```

```
print("Variance of number of installs of Apps on Play Store: " + str(round(var, 2)))
```

```
[ ] # Finding the minimum, maximum, average, median, range, mode, count and variance of the number of installs of all apps
```

```
df = data
maximum = np.max(df['Installs'])
minimum = np.min(df['Installs'])
avg = np.mean(df['Installs'])
med = np.median(df['Installs'])
r = maximum - minimum
var = np.var(df['Installs'])

print("Maximum number of installs of an App on Play Store: " + str(maximum))
print("Minimum number of installs of an App on Play Store: " + str(minimum))
print("Average number of installs of an App on Play Store: " + str(round(avg, 2)))
print("Median number of installs of Apps on Play Store: " + str(round(med, 2)))
print("Range of number of installs of Apps on Play Store: " + str(r))
print("Variance of number of installs of Apps on Play Store: " + str(round(var, 2)))
```

```
Maximum number of installs of an App on Play Store: 1000000000
Minimum number of installs of an App on Play Store: 0
Average number of installs of an App on Play Store: 14172659.72
Median number of installs of Apps on Play Store: 100000.0
Range of number of installs of Apps on Play Store: 1000000000
Variance of number of installs of Apps on Play Store: 6444399933509178.0
```

This data shows us that there exist apps which have no installs.

There is a lot of discrepancy between average and the median value which shows us that the data for number of installs is not increasing linearly.

```
# Determining which Apps are installed the max number of times
data[data['Installs'] == data['Installs'].max()]
```

```
# Determining which Apps are installed the max number of times
data[data['Installs'] == data['Installs'].max()]
```

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres | Last Updated | Current Ver | Minimum Ver |
|-----|--|---------------------|--------|----------|------|------------|------|-------|----------------|-------------------|--------------|--------------------|--------------------|
| 152 | Google Play Books | BOOKS_AND_REFERENCE | 3.9 | 1433233 | 0.0 | 1000000000 | Free | 0.0 | Teen | Books & Reference | 2018-08-03 | Varies with device | Varies with device |
| 335 | Messenger – Text and Video Chat for Free | COMMUNICATION | 4.0 | 56642847 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-08-01 | Varies with device | Varies with device |
| 336 | WhatsApp Messenger | COMMUNICATION | 4.4 | 69119316 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-08-03 | Varies with device | Varies with device |
| 338 | Google Chrome: Fast & Secure | COMMUNICATION | 4.3 | 9642995 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-08-01 | Varies with device | Varies with device |
| 340 | Gmail | COMMUNICATION | 4.3 | 4604324 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-08-02 | Varies with device | Varies with device |
| 341 | Hangouts | COMMUNICATION | 4.0 | 3419249 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-07-21 | Varies with device | Varies with device |
| 382 | Messenger – Text and Video Chat for Free | COMMUNICATION | 4.0 | 56646578 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-08-01 | Varies with device | Varies with device |
| 386 | Hangouts | COMMUNICATION | 4.0 | 3419433 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-07-21 | Varies with device | Varies with device |
| 391 | Skype - free IM & video calls | COMMUNICATION | 4.1 | 10484169 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-08-03 | Varies with device | Varies with device |
| 411 | Google Chrome: Fast & Secure | COMMUNICATION | 4.3 | 9643041 | 0.0 | 1000000000 | Free | 0.0 | Everyone | Communication | 2018-08-01 | Varies with device | Varies with device |

```
# Most reviewed App
data[data['Reviews'] == data['Reviews'].max()]
```

```
# Most reviewed App
data[data['Reviews'] == data['Reviews'].max()]
```

| | App | Category | Rating | Reviews | Size | Installs | Type | Price | Content Rating | Genres | Last Updated | Current Ver | Android Ver |
|------|----------|----------|--------|----------|------|------------|------|-------|----------------|--------|--------------|--------------------|--------------------|
| 2544 | Facebook | SOCIAL | 4.1 | 78158306 | 0.0 | 1000000000 | Free | 0.0 | Teen | Social | 2018-08-03 | Varies with device | Varies with device |

▼ As we can see Facebook is the App that has most number of reviews

```
# Finding the minimum, maximum, average, median, range, mode, count and variance
of the Price of the paid Apps
```

```
df = data[data['Type'] == 'Paid']
maximum = np.max(df['Price'])
minimum = np.min(df['Price'])
avg = np.mean(df['Price'])
med = np.median(df['Price'])
r = maximum - minimum
x, y = sc.mode(df['Price'])
count = len(df)
var = np.var(df['Price'])
```

```
print("Maximum Price of an App on Play Store: $" + str(maximum))
print("Minimum Price of an App on Play Store: $" + str(minimum))
print("Average Price of an App on Play Store: $" + str(round(avg, 2)))
print("Median of Price of Apps on Play Store: $" + str(round(med, 2)))
print("Range of Price of Apps on Play Store: $" + str(r))
print("Variance of Price of Apps on Play Store: $" + str(round(var, 2)))
print("Most occurring Price of App on Play Store (mode): $" + str(x[0]) + " which
occurs " + str(y[0]) + " times")
print("There are " + str(count) + " Paid Apps on Play Store")
```

```
# Finding the minimum, maximum, average, median, range, mode, count and variance of the Price of the paid Apps
df = data[data['Type'] == 'Paid']
maximum = np.max(df['Price'])
minimum = np.min(df['Price'])
avg = np.mean(df['Price'])
med = np.median(df['Price'])
r = maximum - minimum
x, y = sc.mode(df['Price'])
count = len(df)
var = np.var(df['Price'])

print("Maximum Price of an App on Play Store: $" + str(maximum))
print("Minimum Price of an App on Play Store: $" + str(minimum))
print("Average Price of an App on Play Store: $" + str(round(avg, 2)))
print("Median of Price of Apps on Play Store: $" + str(round(med, 2)))
print("Range of Price of Apps on Play Store: $" + str(r))
print("Variance of Price of Apps on Play Store: $" + str(round(var, 2)))
print("Most occurring Price of App on Play Store (mode): $" + str(x[0]) + " which occurs " + str(y[0]) + " times")
print("There are " + str(count) + " Paid Apps on Play Store")
```

```
Maximum Price of an App on Play Store: $400.0
Minimum Price of an App on Play Store: $0.99
Average Price of an App on Play Store: $14.01
Median of Price of Apps on Play Store: $2.99
Range of Price of Apps on Play Store: $399.01
Variance of Price of Apps on Play Store: $3419.68
Most occurring Price of App on Play Store (mode): $0.99 which occurs 144 times
There are 762 Paid Apps on Play Store
```

From the above statistics, we can see that the average Price of paid Apps on the Play Store is closer to the minimum price rather than maximum.

Also the most occurring price is 0.99 dollars which is also the minimum price, hence we can say that most paid apps are inexpensive.

Outcomes:

CO1: Use python libraries like matplotlib, numpy, pandas, scipy for data visualization and scientific/mathematical data computing.

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

Successfully performed exploratory data analysis using Pandas on the database selected.

References:

1. <https://www.geeksforgeeks.org/python-pandas-dataframe/>

