Exploratory Data Analysis (EDA) with Python II

Mentor: Pararawendy Indarjo

Hey I’m,

Pararawendy Indarjo

I am a,

- CURRENTLY | **Senior DS at Bukalapak**
- 19 - 20 | **Data Analyst at Eureka.ai**

BSc Mathematics

MSc Mathematics

Linkedin : https://www.linkedin.com/in/pararawendy-indarjo/
Blog : medium.com/@pararawendy19

Outline

- Beyond standard EDA
  - Deep-dive questions
- Additional techniques
  - Dataframe rows filtering
  - Group-by Aggregation
  - Pivot Table vs Melt
- Hands-on: E-Commerce Dataset
- Q&A

Beyond Standard EDA

- In the previous class, we learned how to extract meaningful insights using standard techniques
  - The methods are directly applicable in most datasets

- In this class, we will enhance our insights by performing **data-deep dive**
  - **By proposing intriguing questions** to be answered by the data at hand

- Why questioning?
  - Questions can be seen as a way to restrict/limit the scope of the analysis
  - Because in BIG data, one can go pointless direction and get drowned by the possibility of the analysis

EDA Steps

1

Data Cleansing

- Column understanding (data dictionary)
- Common “dirt” to clean:
  - Missing data
  - Duplicated data

2

Standard EDA

- Statistical summary of each column
- Univariate analysis
- Multivariate analysis

3

Set deep-dive questions

- Contemplate on the standard EDA findings
  - Are there any meaningful/interesting insights **not** covered yet?
    - AND feasible to answer by the data
- Write down your questions

4

Answer the questions!

- May need to preprocess/manipulate the data first
- Remember: visualize it as much as possible!

Sample Deep-Dive Questions...

- **Aggregation-based**
  - What are the total sales contribution of each product categories?
  - What are Top 10 Products with the most sales?
  - Who are our most loyal/royal customers?
  - Etc...

- **Time series-based**
  - How is the sales trend month-by-month?
  - How is the daily conversion performance for different segments?
  - How is cohort performance? (cohort analysis)
  - Etc...

- **Others**
  - Are purchases from customers from region A is higher in value compared to the ones from region B?
  - How is the profile of our users? (user segmentation)
  - Etc...

You'll get the intuition on "what to ask?" as you gain <u>more</u> experiences

Any

Question

Data frame Filtering

Additional Technique #1

Rows Filtering

General syntax format

df[filter_condition]

For example, we want to filter only rows
column_name with value XXX

Then filter_condition follows the below pattern :

df['column_name'] == XXX

Full code to filter:

df[df['column_name'] == XXX]

Rows Filtering

General syntax format

df[filter_condition]

For example, we want to filter only rows
column_name with value XXX

Then filter_condition follows the below pattern :

df['column_name'] == XXX

Full code to filter:

df[df['column_name'] == XXX]

Remember Boolean-Indexing?

Rows Filtering

E.g. we want to retrieve female passengers data

```
df[df['sex']=='female']
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
8	1	3	female	27.0	0	2	11.1333	S	Third	woman	False	NaN	Southampton	yes	False
9	1	2	female	14.0	1	0	30.0708	C	Second	child	False	NaN	Cherbourg	yes	False
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True
14	0	3	female	14.0	0	0	7.8542	S	Third	child	False	NaN	Southampton	no	True
15	1	2	female	55.0	0	0	16.0000	S	Second	woman	False	NaN	Southampton	yes	True
18	0	3	female	31.0	1	0	18.0000	S	Third	woman	False	NaN	Southampton	no	False

Filtering Types

1. Logical operators
2. Isin, Isnull, notnull
3. Tilde (~)

Essentially Boolean-Indexing

Logical operators

Pandas Code	SQL Syntax
df[(df['column'] == a) "&"/"|" (df['column'] == b)]	where <column> = a "and"/"or" <column> = b
Example : Data of passengers older than 30 y.o (and/or) fare greater than $20	

```
df[(df['age']>30)&(df['fare']>20)]
df[(df['age']>30)|(df['fare']>20)]
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False	NaN	Southampton	no	False
9	1	2	female	14.0	1	0	30.0708	C	Second	child	False	NaN	Cherbourg	yes	False
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True
13	0	3	male	39.0	1	5	31.2750	S	Third	man	True	NaN	Southampton	no	False
15	1	2	female	55.0	0	0	16.0000	S	Second	woman	False	NaN	Southampton	yes	True
16	0	3	male	2.0	4	1	29.1250	Q	Third	child	False	NaN	Queenstown	no	False

Isin, Isnull, notnull

Pandas Code	SQL Syntax
df[df['column'].isin([a,b,c])] df[df['column'].isnull()] df[df['column'].notnull()]	where <column> in (a,b,c) where <column> is null where <column> is not null

Cases :

1. [isin] - Retrieve data when the value is contained in a certain value reference (list)
2. [isnull] - Retrieve data when the value is null
3. [notnull] - Retrieve data when the value is NOT null

Isin, Isnull, notnull

Pandas Code	SQL Syntax
df[df[‘column’].isin([a,b,c])] df[df[‘column’].isnull()] df[df[‘column’].notnull()]	where <column> in (a,b,c) where <column> is null where <column> is not null

Kasus :

1. [isin] `df[df['class'].isin(['First','Second'])]`
2. [isnull] `df[df['age'].isnull()]`
3. [notnull] `df[df['deck'].notnull()]`

Tilde (~)

Pandas Code	SQL Syntax
df[~df['column'].str.contains('South')] df[~df['column'].isin([a,b,c])]	where <column> **not** like '%South%' where <column> **not** in (a,b,c)

Tilde is negation (opposite boolean value)

Cases:

1. [~contains] - retrieve data when value does **not** contain a certain string pattern
2. [~isin] - retrieve data when value is **not** included in a certain value reference

Tilde (~)

Pandas Code	SQL Syntax
df[~df[‘column’].str.contains('South’)] df[~df[‘column’].isin([a,b,c])]	where <column> **not** like ‘%South%’ where <column> **not** in (a,b,c)

Cases :

1. [~contains] `df[~df[‘embark_town’].str.contains('South')]` → NOT contain ‘South’
2. [~isin] :`df[~df[‘deck’].isin(['C',’E’])]` → NEITHER ‘C’ or ‘E’

Hands-On

- Open today's Jupyter notebook on your Google Colab!
- Make sure you have uploaded the required CSV files to your google drive
    - Remember the file path!

Exercise

Retrieve the data of passengers max age at 30 y.o located in the first class

Code ???

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	E	Southampton	no	True
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True
30	0	1	male	40.0	0	0	27.7208	C	First	man	True	NaN	Cherbourg	no	True
35	0	1	male	42.0	1	0	52.0000	S	First	man	True	NaN	Southampton	no	False
52	1	1	female	49.0	1	0	76.7292	C	First	woman	False	D	Cherbourg	yes	False
54	0	1	male	65.0	0	1	61.9792	C	First	man	True	B	Cherbourg	no	False
61	1	1	female	38.0	0	0	80.0000	NaN	First	woman	False	B	NaN	yes	True
62	0	1	male	45.0	1	0	83.4750	S	First	man	True	C	Southampton	no	False

Exercise

Dataframe: titanic

Retrieve survived passengers who didn't embark from Queenstown

Code ???

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
8	1	3	female	27.0	0	2	11.1333	S	Third	woman	False	NaN	Southampton	yes	False
9	1	2	female	14.0	1	0	30.0708	C	Second	child	False	NaN	Cherbourg	yes	False
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False	G	Southampton	yes	False
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	C	Southampton	yes	True
15	1	2	female	55.0	0	0	16.0000	S	Second	woman	False	NaN	Southampton	yes	True
17	1	2	male	NaN	0	0	13.0000	S	Second	man	True	NaN	Southampton	yes	True
19	1	3	female	NaN	0	0	7.2250	C	Third	woman	False	NaN	Cherbourg	yes	True

Any

Question

Group-by aggregation

Additional Technique #2

Aggregation

- Grouping data based on certain Columns, then perform some aggregate operation on the grouped data.
- Group by aggregation is useful especially when we have granular data, and we want to create more compact representation of it (data summarization)
    - E.g. transaction data

Format Code :

New aggregated column name

```
df.groupby([ 'basecol' ]).agg(
                    namecol1=( 'aggcol1' ,'aggfunction' ),
                    namecol2=( 'aggcol2' ,'aggfunction' ),
                    . . .
                  )
```

Basis group-by column

Target column to aggregate

Aggregating function name

Aggregation

List of aggregate functions

- count() – Number of non-null observations
- nunique() – Number of distinct values
- sum() – Sum of values
- mean() – Mean of values
- median() – Median of values
- min() – Minimum
- max() – Maximum
- mode() – Mode
- std() – Standard deviation
- var() – Variance

Sample problem to solve by group-by aggregation:
How is the average of ticket fares per gender?

```
(titanic.groupby(['sex'])
         .agg(avg_fare=('fare','mean')))
```

	avg_fare
sex	
female	44.479818
male	25.523893

Aggregation

Data `titanic`

Case: finding the oldest passengers & the most expensive ticket price based on survival status.

Code :

```
titanic.groupby(['survived']).agg( {
                                max_age=('age','max'),
                                  max_fare=('fare','max')
                              })
```

	max_age	max_fare
survived		
0	74.0	263.0000
1	80.0	512.3292

Aggregation - Reset Index

- After aggregating, the base group-by column will become the index of the resulting dataframe.
- To make it as a column, use `reset_index` method

```
titanic.groupby('survived').agg(
    max_age = ('age','max'),
    max_fare = ('fare','max')
)
```

survived	max_age	max_fare
0	74.0	263.0000
1	80.0	512.3292

```
titanic.groupby('survived').agg(
    max_age = ('age','max'),
    max_fare = ('fare','max')
).reset_index()
```

	survived	max_age	max_fare
0	0	74.0	263.0000
1	1	80.0	512.3292

Hands-On

- Open today's Jupyter notebook on your Google Colab!
- Make sure you have uploaded the required CSV files to your google drive
    - Remember the file path!

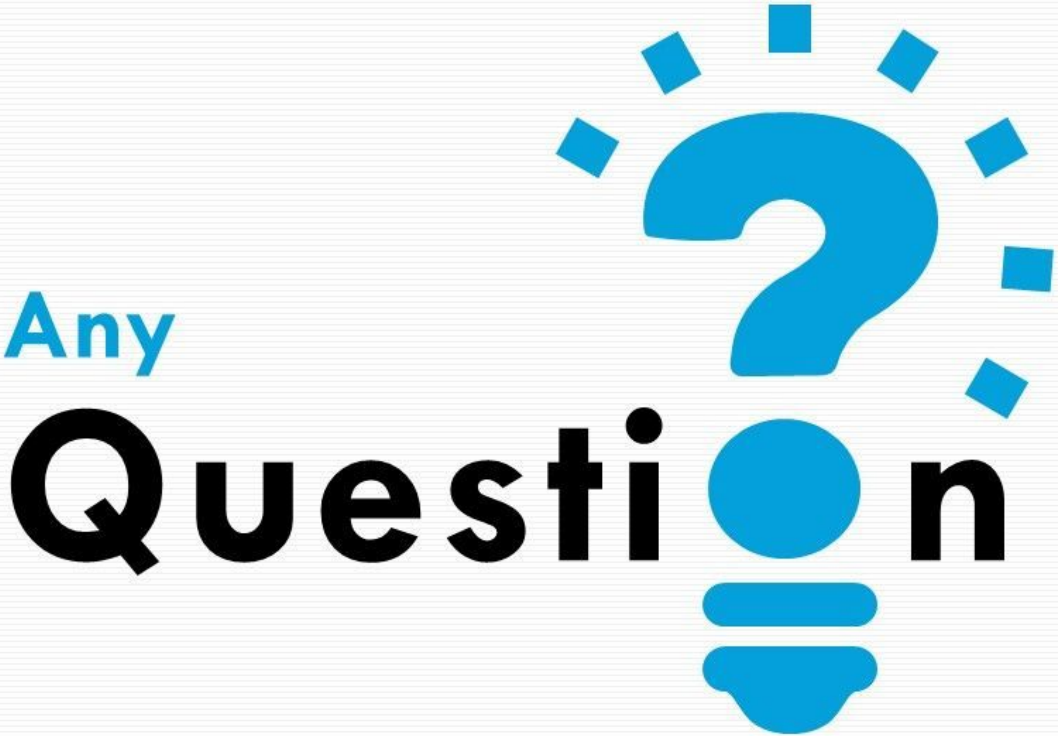
Exercise

Dataframe: `titanic`

Reproduce the following aggregated dataframe!

Code ???

	sex	survived	median_fare	avg_age
0	female	0	15.24580	25.046875
1	female	1	26.00000	28.847716
2	male	0	9.41665	31.618056
3	male	1	26.28750	27.276022


Any
Question

Pivot Table vs Melt

Additional Technique #3

Pivot Table

- Pivot table allows us to create a **spreadsheet-style pivot table** as a DataFrame

- This is particularly useful if we have “long” formatted dataframe
  - E.g. Resulting dataframe from group-by aggregation!

- General syntax format

```
pd.pivot_table(dataframe,
               index,          → Column(s) set as index (rows)
               columns,        → Column(s) set as column
               values,         → (Optional) Column(s) whose values to show
               aggfunc='mean') → Aggregating function (if >1 values to show)
```

Pivot Table

- Suppose we have the following dataframe

	sex	sibsp	median_fare
0	female	0	13.0000
1	female	1	28.4500
2	female	2	27.0000
3	female	3	25.4667
4	female	4	31.2750
5	female	5	46.9000
6	female	8	69.5500
7	male	0	8.0500
8	male	1	26.0000
9	male	2	23.2500
10	male	3	27.9000
11	male	4	31.3875
12	male	5	46.9000
13	male	8	69.5500

Pivot

```
df_pivot = pd.pivot_table(df,
              index='sex',
              columns='sibsp')
df_pivot
```

	median_fare						
sibsp	0	1	2	3	4	5	8
sex							
female	13.00	28.45	27.00	25.4667	31.2750	46.9	69.55
male	8.05	26.00	23.25	27.9000	31.3875	46.9	69.55

Melt

- Melt is the **reverse** of pivot table
    - I.e. you have "wide" formatted dataframe and want to make it "long"

- This is sometimes useful to prepare data for visualization
    - Which requires "long" formatted data, e.g. seaborn package

- General syntax format

```
df.melt(id_vars,        →  Column(s) that remain as index (rows)
        var_name,       →  Resulting column name of melted columns
        value_name      →  Name of the values column
        )
```

Melt

- Suppose we have the following dataframe

	sex	0	1	2	3	4	5	8
0	female	13.00	28.45	27.00	25.4667	31.2750	46.9	69.55
1	male	8.05	26.00	23.25	27.9000	31.3875	46.9	69.55

```
df.melt(id_vars = 'sex',
        var_name='sibsp',
        value_name='median_fare')
```

Melt

	sex	sibsp	median_fare
0	female	0	13.0000
1	male	0	8.0500
2	female	1	28.4500
3	male	1	26.0000
4	female	2	27.0000
5	male	2	23.2500
6	female	3	25.4667
7	male	3	27.9000
8	female	4	31.2750
9	male	4	31.3875
10	female	5	46.9000
11	male	5	46.9000
12	female	8	69.5500
13	male	8	69.5500

Hands-On

- Open today's Jupyter notebook on your Google Colab!
- Make sure you have uploaded the required CSV files to your google drive
    - Remember the file path!

Exercise

Base Dataframe code

```
titanic.groupby(['embark_town','class','parch']).agg(
    avg_age = ('age','mean')
).reset_index()
```

Code ???

Reproduce the following dataframe!

Hint: shown numbers are average of ages

class	First	Second	Third
embark_town			
Cherbourg	35.750314	23.233333	18.941364
Queenstown	38.500000	43.500000	27.784314
Southampton	41.354403	29.203384	29.930145

Any

Question

Hands-On

- Open today's Jupyter notebook on your Google Colab!
- Make sure you have uploaded the required CSV files to your google drive
    - Remember the file path!

Thank you

Assignment

- Dataset : https://www.kaggle.com/blastchar/telco-customer-churn
- What to submit? Google colab link (don't forget to share access to me: pararawendy19@gmail.com)
    - Format notebook name: HW_EDAII_<YOUR COMPLETE NAME>
- First data preprocessing code after loading the data as `df` variable:

```python
# exclude rows with TotalCharges column contains white space
df = df.loc[~df['TotalCharges'].str.contains(' ')]

# transform TotalCharges col to float
df['TotalCharges'] = df['TotalCharges'].astype(float)
```

Instructions

- Perform standard data cleansing **(10 points)**
    - Missing values
    - Duplicated values
- Perform standard EDA with rich interpretations! **(70 points)**
    - Statistical summary of columns (20 points)
    - Univariate analysis (20 points)
    - Multivariate analysis (30 points)
- Perform deep-dive exploration **(20 points)**
    - Ask minimum 2 questions
    - At least 1 of them should involve group-by aggregation!