Exploratory Data Analysis (EDA) with Python I

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

Universiteit Leiden

MSc Mathematics

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

Outline

- What is Exploratory Data Analysis (EDA)?
  - And why we need it?
- Data Cleansing
- Statistical Summaries
- Univariate Analysis
- Multivariate Analysis
- Hands-On

What is Exploratory Data Analysis (EDA)?

- An exercise to explore the data, **<u>to uncover insights</u>** from it
  - I.e. to really understand the data

- How exactly to do the “exploration”?
  - Essentially by looking at various angles of the data

- In today’s session, we will learn several **standard techniques** to perform EDA

Why doing EDA?
3 Objectives

1

Data Cleansing

- Real world data is messy
- Common “dirt” to clean:
  - Missing data
  - Duplicated data

2

Data Understanding

- Essence of EDA
  - Explore data to get insights
  - What is the data telling us?
- Statistical summary of each column
- Univariate analysis
- Multivariate analysis

3

Model Feature Selection*

- *This is optional: when we want to build a model
- EDA can detect
  - Potential promising features
  - Redundant features
    - Keep only one of them

Dataset used throughout material session

MPG dataset

- MPG (mile per gallon) data is a dataset contains cars' specification details
- This dataset can be loaded from Seaborn package

```
mpg = sns.load_dataset('mpg')
mpg.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

Data Cleansing

There are two aspects

01

Missing Values

- Missing values are quite common in real world data
- We can drop rows if they contain missing values, as long as the number is relatively small (< 5% of total rows)
- If so dominant, push back to data owner (Data Engineer) OR ignore the column altogether

02

Duplicated Rows

- Duplicated rows can happen because of:
  - Double inputs error
  - Inappropriate SQL JOINS
- We need to drop them, since we don't want they make our analysis bias towards duplicated rows

Missing Values

How to check them

df.info()

```
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    object
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

- df.info() is a simple but powerful method
- It shows many information of the dataframe
    - Number of rows
    - Columns' information:
        - Name
        - Number of non null (non-missing) values
        - Data type

Missing Values

How to check them

df.info() ← **Check them** → df.isna().sum()

```
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    object
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

```
mpg.isna().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
name            0
dtype: int64
```

Missing Values

How to check and drop them afterwards (if any)

Check them

df.info()

```
mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    int64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    object
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

df.isna().sum()

```
mpg.isna().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
name            0
dtype: int64
```

Drop them

df = df.dropna()

```
mpg = mpg.dropna()
```

```
# check after dropping
mpg.isna().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model_year      0
origin          0
name            0
dtype: int64
```

Missing Values

Some notes

- There is another strategy to handle missing values, called Imputation

- Imputation: replacing missing values with some values **based on our assumption**

- Typical strategy:
    - Numerical column: replace with median
    - Categorical column: replace with mode

- BUT, if the objective is EDA, imputation is NOT recommended
    - Because it can lead to a biased insights/conclusion

- So, only use imputation in **preparing modelling dataset**

Duplicated Values

How to check and drop them afterwards (if any)

- To check duplicated rows, the syntax is
  - `df.duplicated().sum()`

```
mpg.duplicated().sum()

0
```

- Turns out our mpg data does NOT have duplicated row (which is good)

- IN CASE there are duplicated rows, we drop them using the following syntax
  - `df = df.drop_duplicates()`


Any
Question

Statistical Summary of Columns

- The first thing to do is to **look at the statistical summary** of each column
- Objectives: to understand **how the values are distributed** for each column
  - What are the min/max values? Do they make sense?
  - Is the distribution skewed or symmetric?
    - Median != Mean → Skewed (Not symmetric)
    - Median ~ Mean → Symmetric
  - How is each value's frequency? (for categorical column)
- Best practice: separate your column names!

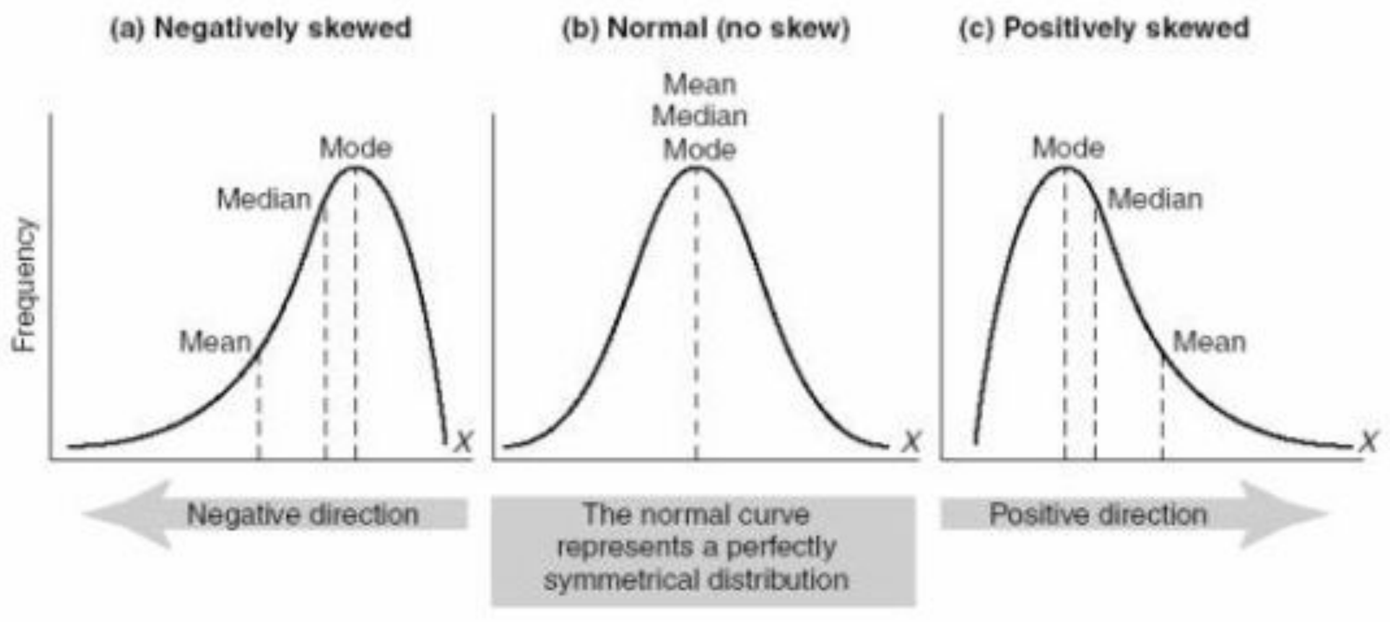
```
cats = ['origin', 'name']
nums = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
        'acceleration', 'model_year']
```

Review: Quartiles

Q1, Q2, Q3 values are important statistics of sorted data

Review: Distribution Forms

There are three forms of distribution



Numerical Columns

Syntax: df[nums].describe()

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

- Minimum and maximum values for all columns seemed reasonable

Numerical Columns

Syntax: df[nums].describe()

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

- Minimum and maximum values for all columns seemed reasonable
- displacement has skewed distribution (mean >> median)
- acceleration has roughly symmetrical distribution (mean ~ median)

Categorical Columns

Syntax: df[cats].describe()

	origin	name
count	398	398
unique	3	305
top	usa	ford pinto
freq	249	6

- For categorical columns, the method will output
    - the number of unique/distinct categories within the column
    - information about the most frequent category in the column

- In our mpg dataset:
    - There are three cars' origins, and USA is the most common one
    - There are so many car names (305 unique names!).
        - We **may neglect** this column for further analysis

Categorical Columns

df[‘column_name’].value_counts()

```
mpg['origin'].value_counts()
```

```
usa       249
japan      79
europe     70
Name: origin, dtype: int64
```

- We can further inspect the exact frequency of each value of a categorical column.
- From the left figure:
    - Turns out Japan is the second top car producent
    - And Europe cars are the least in number

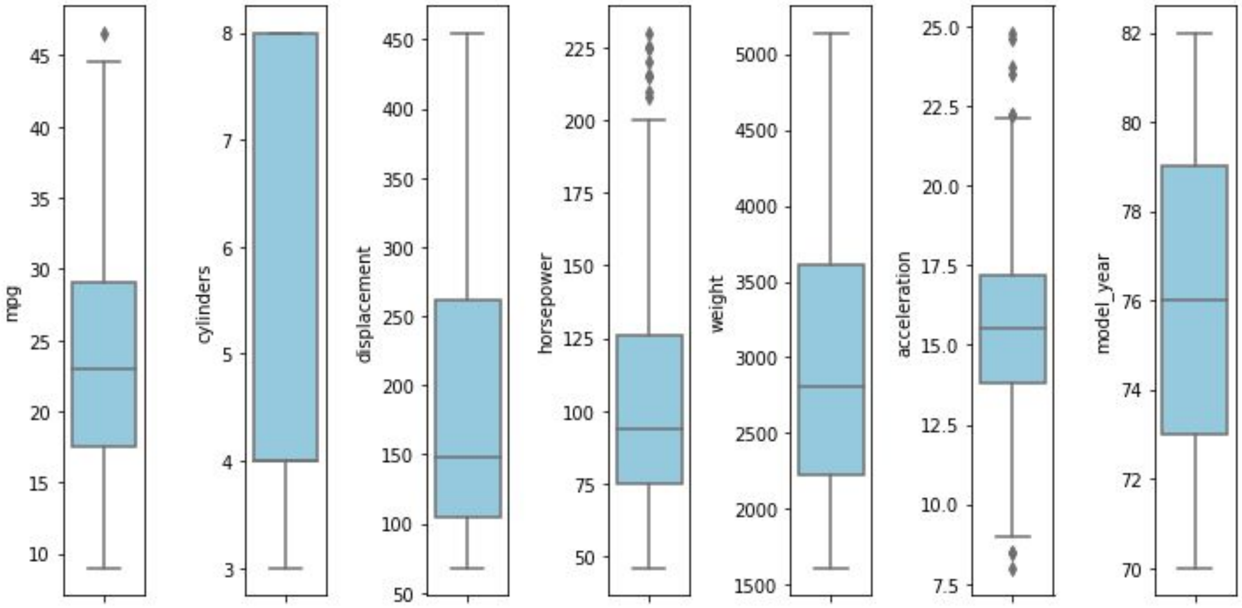
Any Question

Univariate Analysis

- Univariate analysis is an exercise to **visually analyze each column one-by-one**
- So that we can improve our understanding towards them
- In the following, we will learn to
  - Detect outliers using boxplot
  - Inspect/validate the distribution form by plotting histogram/KDE (kernel density estimation) plot

Detect Outliers via Boxplot

```
features = nums
plt.figure(figsize=(10,5))
for i in range(0, len(features)):
    plt.subplot(1, len(features), i+1)
    sns.boxplot(y=mpg[features[i]], color='skyblue')
    plt.tight_layout()
```



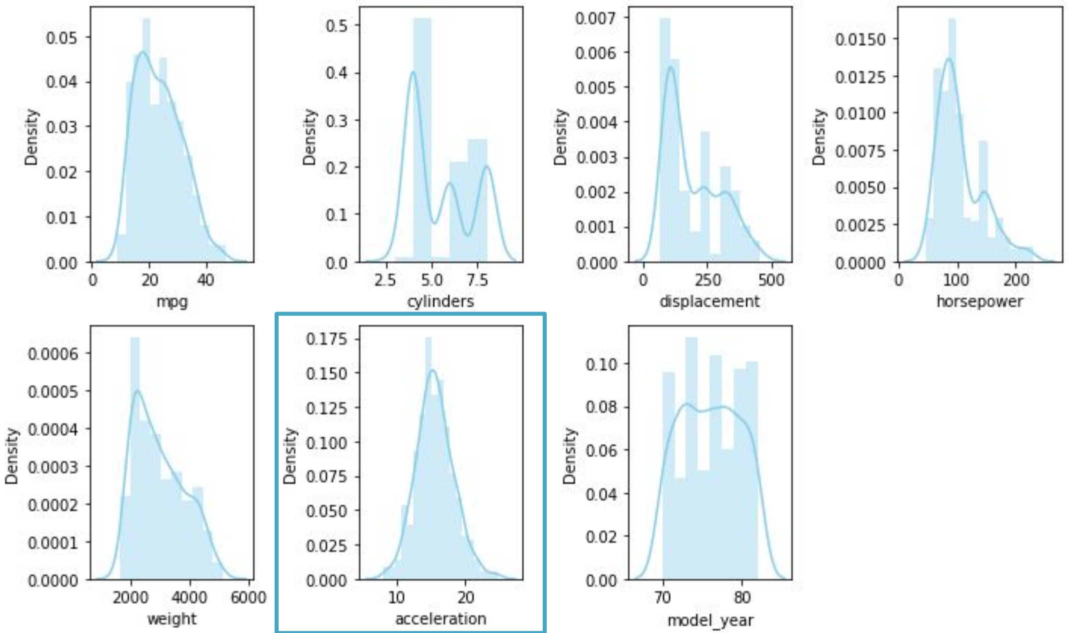
- Column mpg, horsepower, and acceleration have handful of outliers

- That said, the outliers are still "normal" (not too extreme)
  - So they seemed to be valid data points
  - I.e. no need to drop

- IF the data contains non-sensical values, drop them using boolean indexing
  - E.g. kolom 'umur' dengan values > 200 tahun

Inspect Column Distribution

```python
features = nums
plt.figure(figsize=(10,6))
for i in range(0, len(features)):
    plt.subplot(2, 4, i+1)
    sns.distplot(x=mpg[features[i]], color='skyblue')
    plt.xlabel(features[i])
    plt.tight_layout()
```



- Most cars are (distribution peak)
  - ~17 miles/gallon
  - 4 cylinders
  - ~100 disp
  - ~95 horsepower
  - ~2200 kg in weights
  - ~15 in acceleration

- Acceleration has the most symmetric distribution

- Many columns are positively skewed:
  - mpg
  - displacement
  - horsepower
  - weight

Inspect Column Distribution

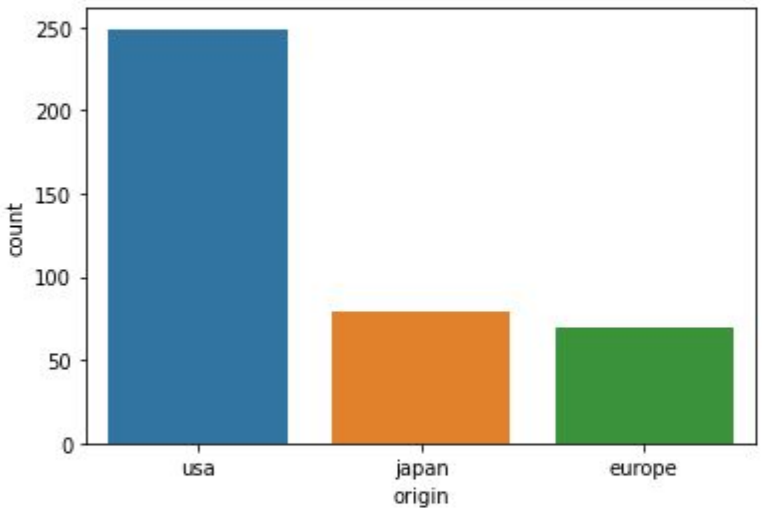
Some notes

- In case of discrete column with not many distinct values, no need to conclude it's symmetricity
  - Just focus on the balance-level of how values are distributed

- If your end objective is to perform modelling:
  - Beware of imbalance target variable distribution (in binary classification)
  - May perform LOG transformation for columns with highly positive skew
    - Hopefully to better approximate normal distribution

Countplot for Categorical Columns

```
sns.countplot(data=mpg, x='origin')
```



- Essentially a visual version of **value_counts()** method we did earlier
- Most of cars are form USA, followed by Japan and Europe with roughly the same number of cars

Any

Question

Multivariate Analysis

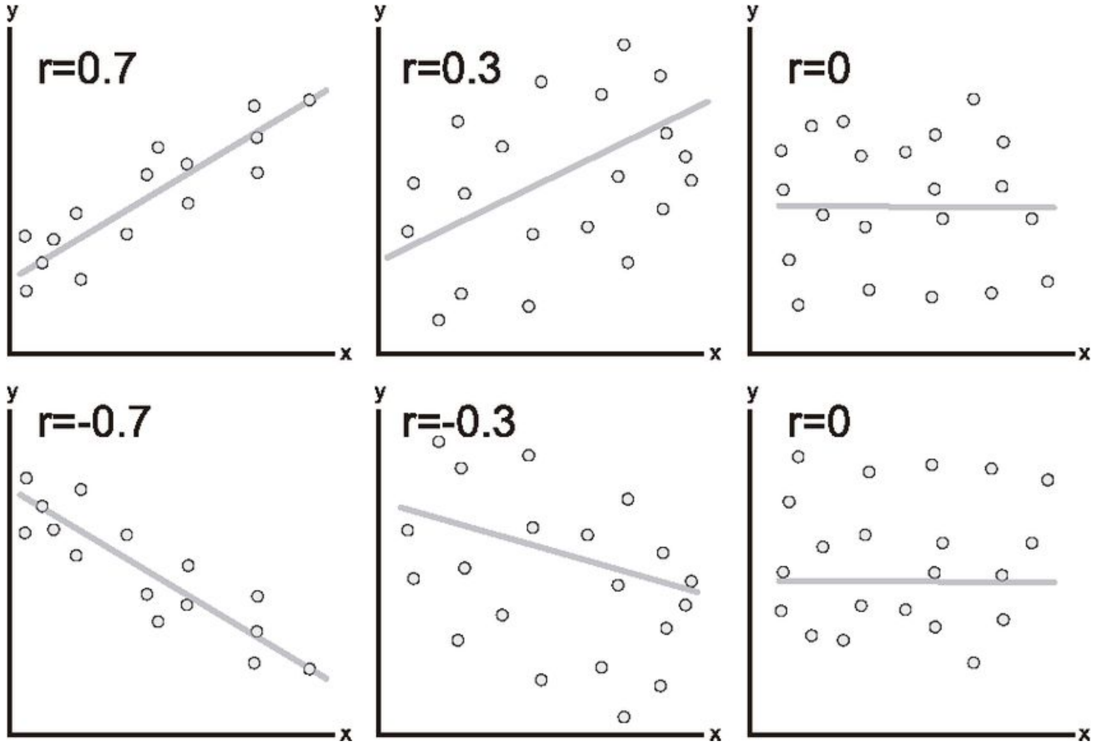
- In multivariate analysis, we **analyze multiple variables all at once**
- So that we can **<u>understand their relationship</u>**
- In the following, we will learn to
    - See the Pearson's correlation values between column pairs
    - Draw and interpret pairplot

Review: Pearson's Correlation

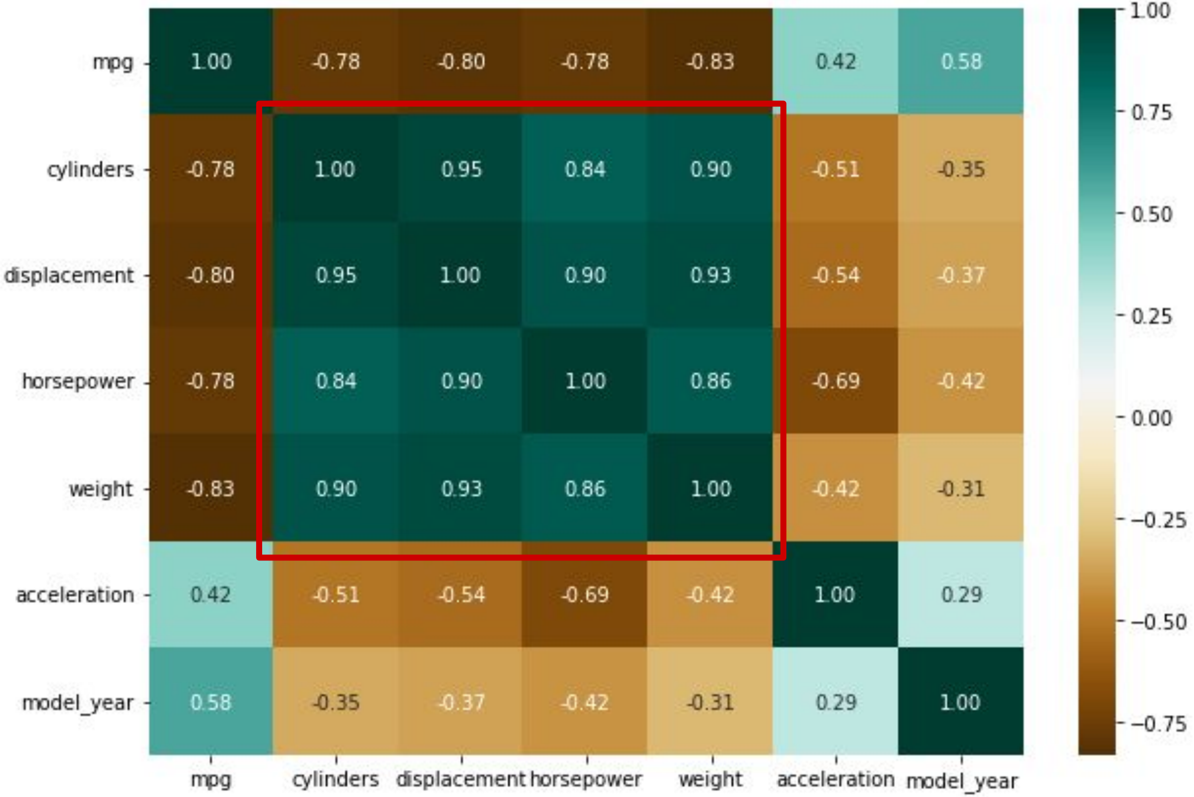
Measures what???

Review: Pearson’s Correlation

Measures LINEAR relationship between two variables

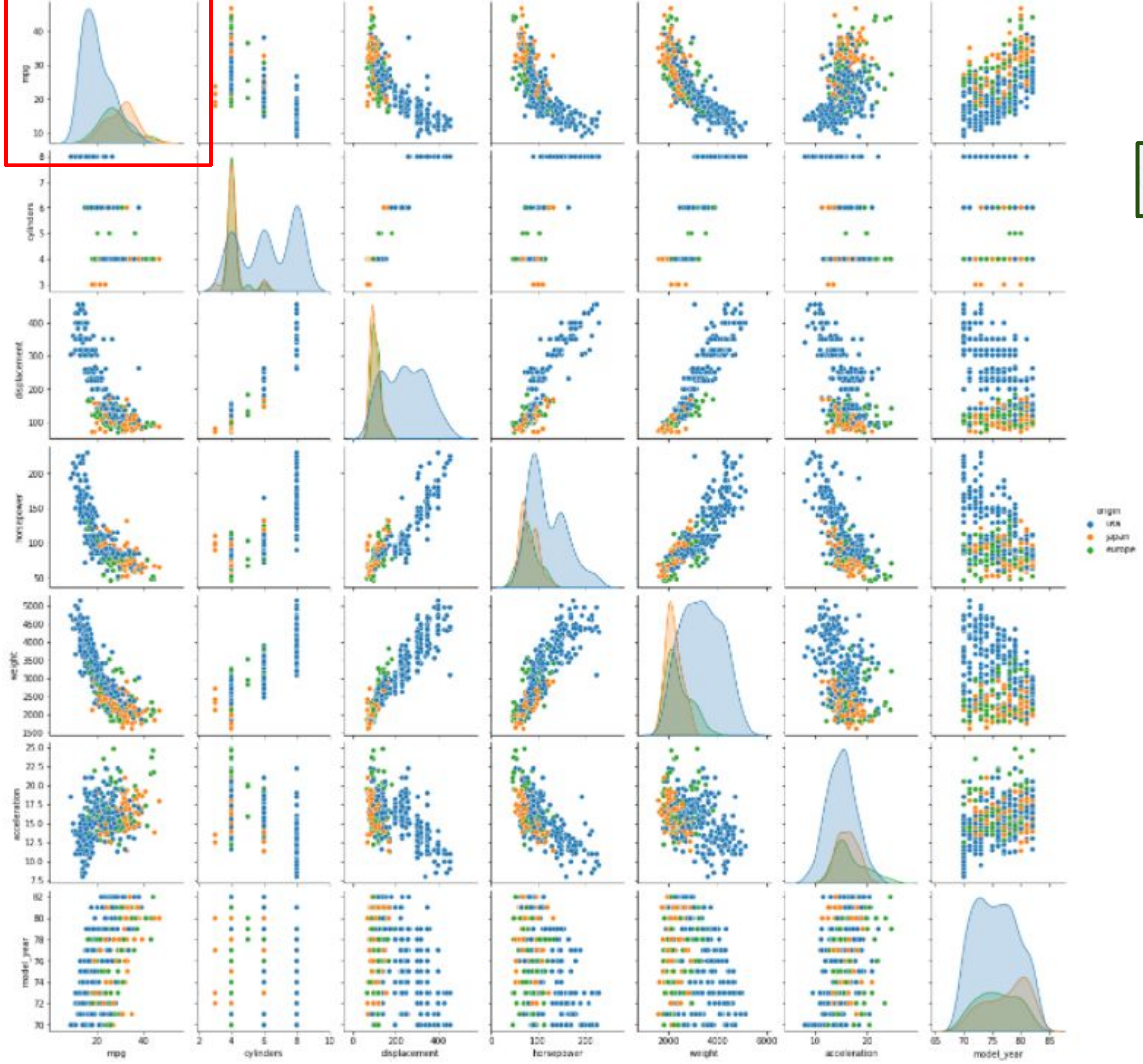


Correlation Heatmap




```
correlation = mpg.corr()
plt.figure(figsize=(10,7))
sns.heatmap(correlation, annot=True,
            fmt='.2f', cmap='BrBG')
```

- Variables inside red rectangle are highly correlated each other
- This means they contain redundant information
  - I.e. we can choose only 1 of them to modelling process
  - Will be explained further in Regression classes



Pairplot

```
sns.pairplot(mpg, hue='origin')
```

- Legend: **Origin**
  - usa
  - japan
  - europe
- Usa cars have more spread distribution nearly in all columns
- Notice regarding mpg (most left in the plot), usa cars are **less efficient in mile per gallon**

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