Data Analysis With Python

Dr. Amar Singh
Professor, School Of Computer Applications
Lovely Professional University

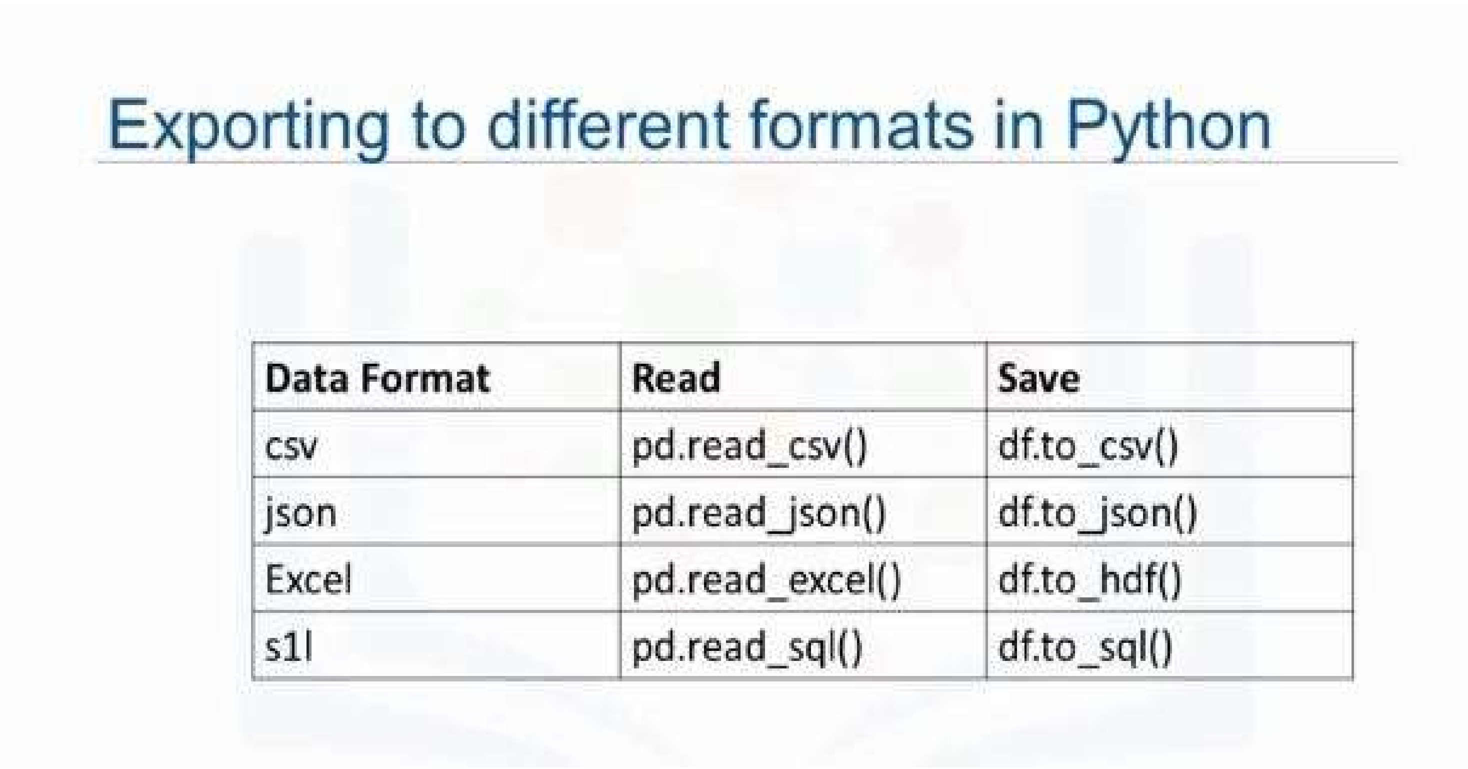
Libraries in Python

- Scientific Computing Libraries
  - NumPy
  - Pandas
- Visualization Libraries
  - Matplotlib
  - Seaborn
- Algorithmic Libraries
  - Scikit-learn

Importing Data in Python

- Importing is the process of loading or reading the data from different resources.
- The data may be in different formats.
    - .csv, .json, .xlsx
- Path of the dataset could be mentioned as below:
    - C:\\mydata\\data.csv
- To read a csv file we can use following command:
    - pd.read_csv(“c:\\mydata\\data.csv”)

Libraries



Exporting to different formats in Python

Data Format	Read	Save
csv	pd.read_csv()	df.to_csv()
json	pd.read_json()	df.to_json()
Excel	pd.read_excel()	df.to_hdf()
s1l	pd.read_sql()	df.to_sql()

Check data type

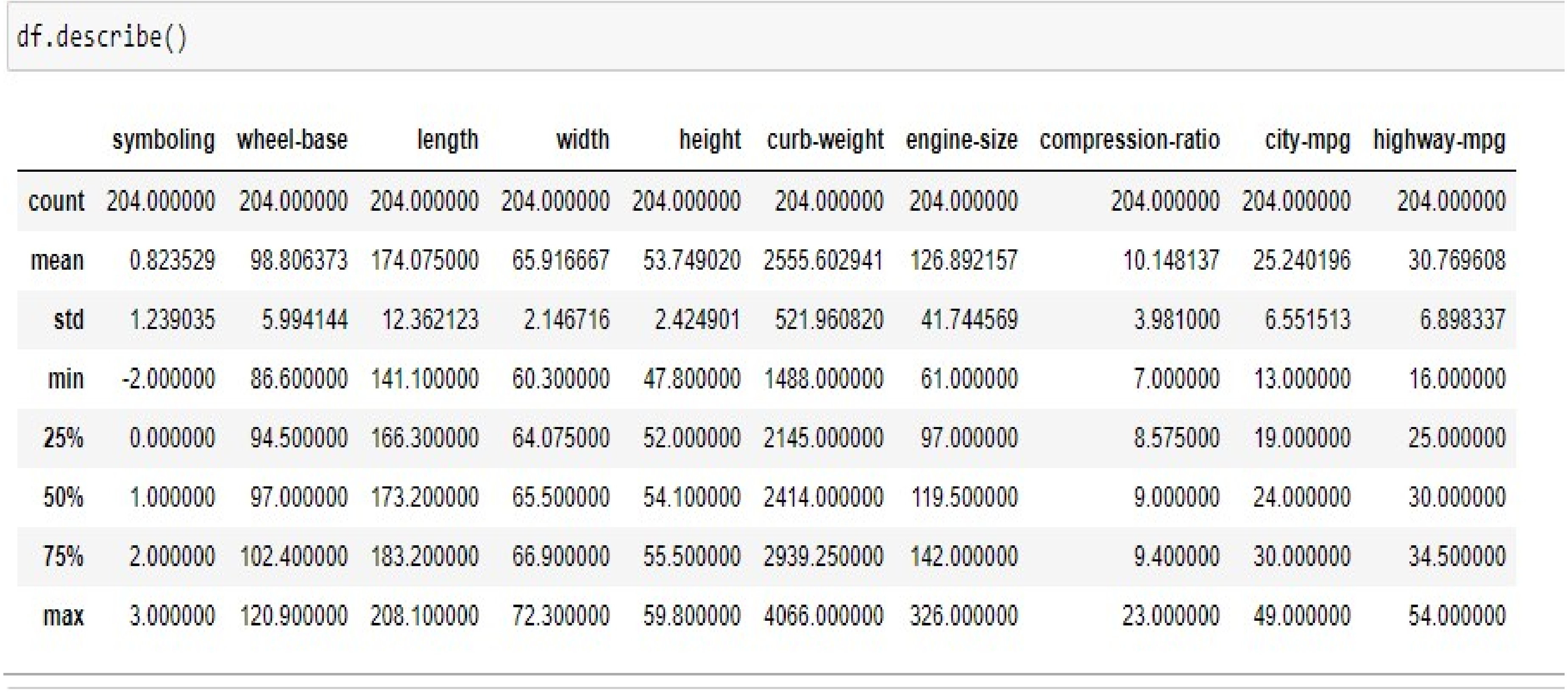
- Dataframe.dtypes

Printing Dataframe

- df[“BasePay”] // Prints only BasePay column
- df.head(n) //shows first n rows of the data frame
- df.tail(n) //shows bottom n rows of the data frame
- Df.dtypes // used to check data types

Dataframe.describe()

- Returns full summary Statistical



```
df.describe()
```

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	204.000000	204.000000	204.000000	204.000000	204.000000	204.000000	204.000000	204.000000	204.000000	204.000000
mean	0.823529	98.806373	174.075000	65.916667	53.749020	2555.602941	126.892157	10.148137	25.240196	30.769608
std	1.239035	5.994144	12.362123	2.146716	2.424901	521.960820	41.744569	3.981000	6.551513	6.898337
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.075000	52.000000	2145.000000	97.000000	8.575000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	119.500000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.200000	66.900000	55.500000	2939.250000	142.000000	9.400000	30.000000	34.500000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

Data Pre-processing

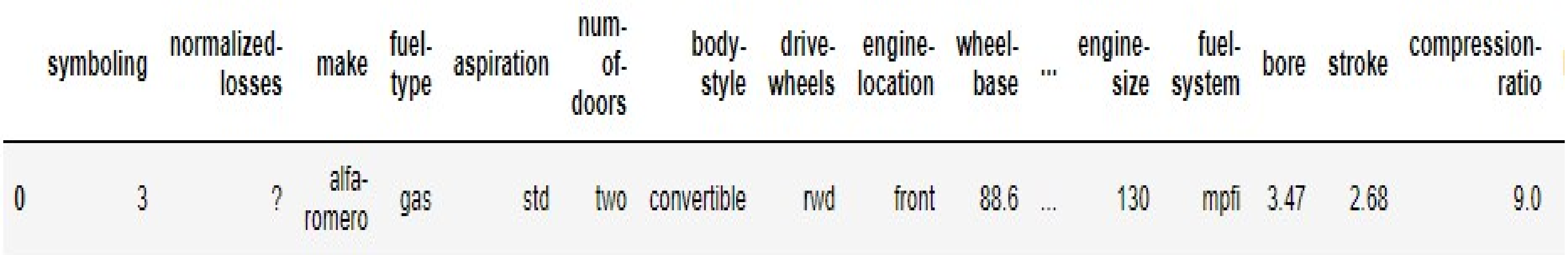
- Pre-Processing is used to convert raw data into another format for further data analysis.
- Also known as data cleaning or data wrangling.

Data-Preprocessing

- Deal with missing values
- Data Formatting
- Data Normalization
- Converting Categorical Values to Numerical Values

Missing Values

- When no value is stored for column in an observation.
- Could be represented as ?, NA or blank cell.



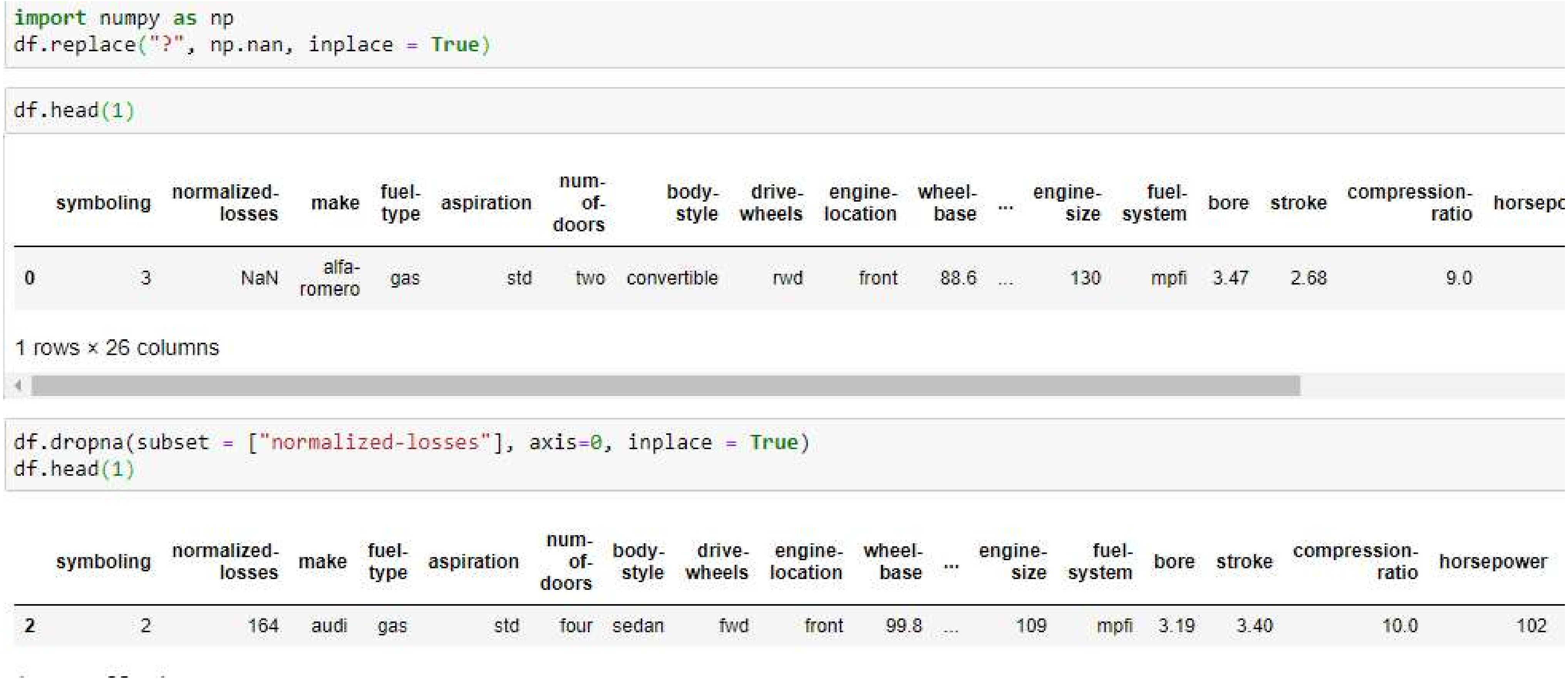
	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0

How to deal with missing data

- Drop missing values
  - Drop the variable
- Replace missing values with an average or frequency values.
- Leave it as missing data.

How to drop missing values in python

- Use dataframe.dropna()



```
import numpy as np
df.replace("?", np.nan, inplace = True)
```

```
df.head(1)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepc
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	

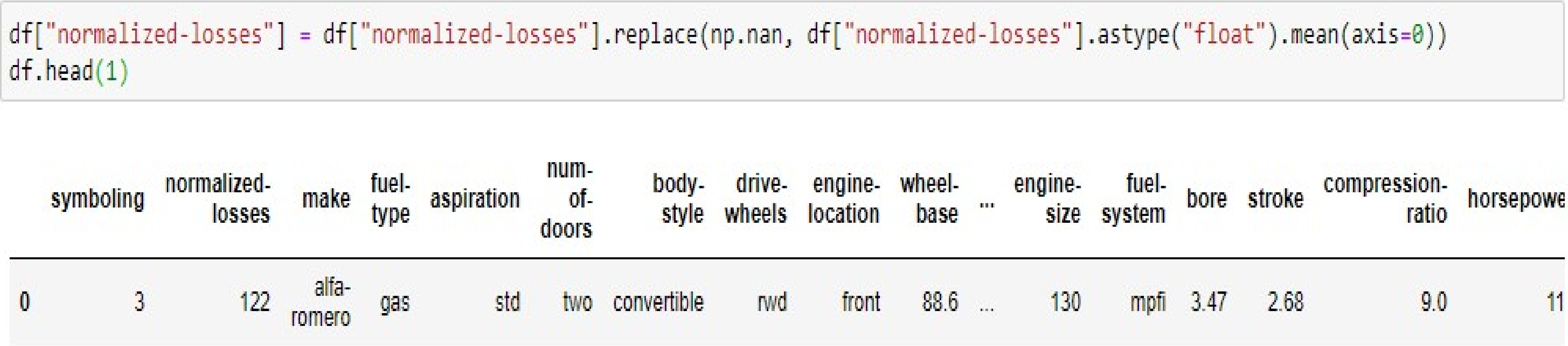
1 rows × 26 columns

```
df.dropna(subset = ["normalized-losses"], axis=0, inplace = True)
df.head(1)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102

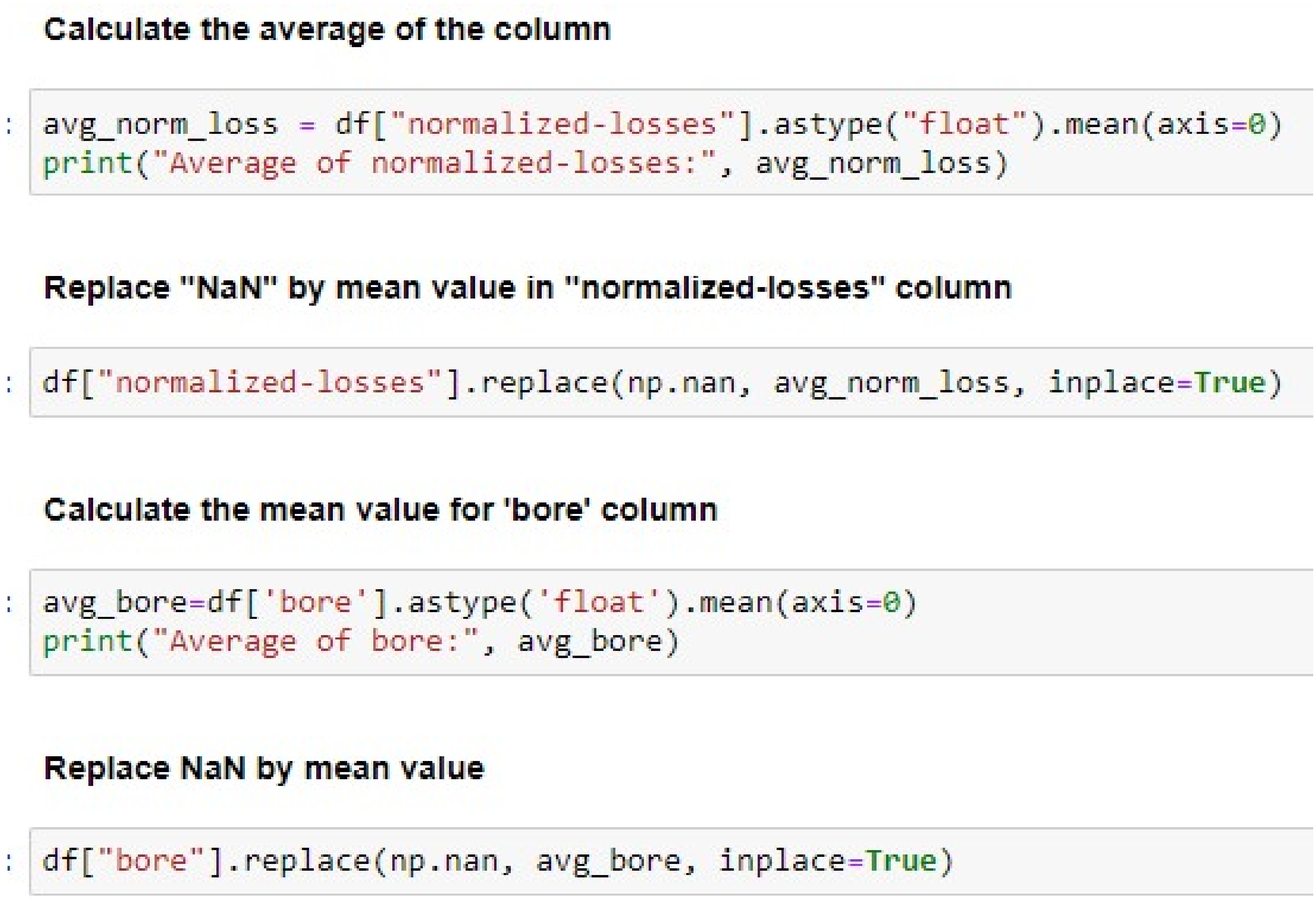
How to replace missing value with new value ?

- Df.replace(missing value, new value)



```
df["normalized-losses"] = df["normalized-losses"].replace(np.nan, df["normalized-losses"].astype("float").mean(axis=0))
df.head(1)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepowe
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	11



Calculate the average of the column

```
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-losses:", avg_norm_loss)
```

Replace "NaN" by mean value in "normalized-losses" column

```
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
```

Calculate the mean value for 'bore' column

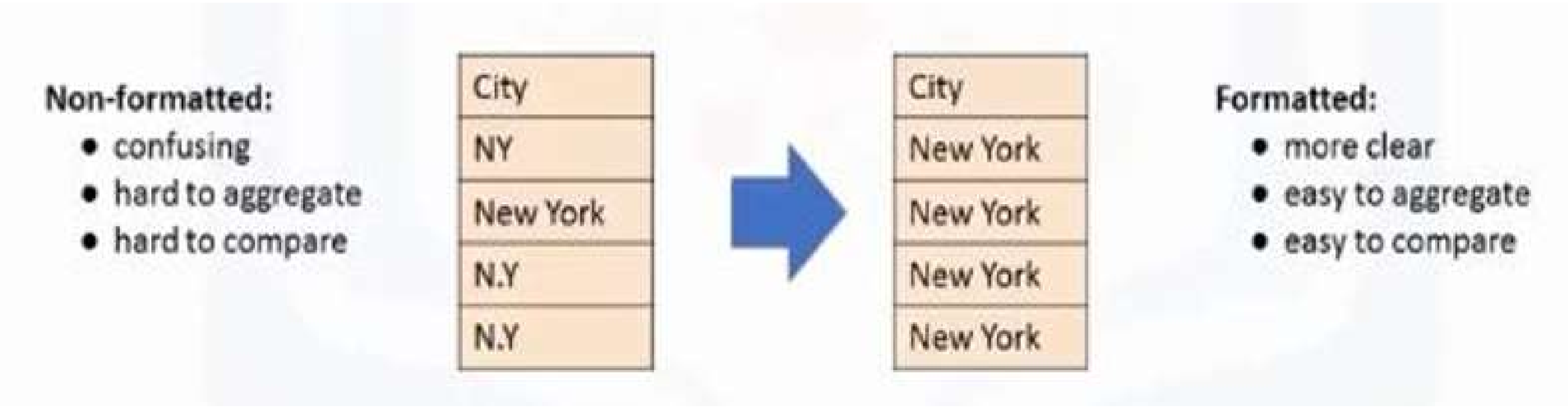
```
avg_bore=df['bore'].astype('float').mean(axis=0)
print("Average of bore:", avg_bore)
```

Replace NaN by mean value

```
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

Data Formatting

- Data are usually collected from different sources and stored in different formats.
- Bringing data into standard of expression allows user to make meaningful comparisons.



Non-formatted:

- confusing
- hard to aggregate
- hard to compare

City
NY
New York
N.Y
N.Y

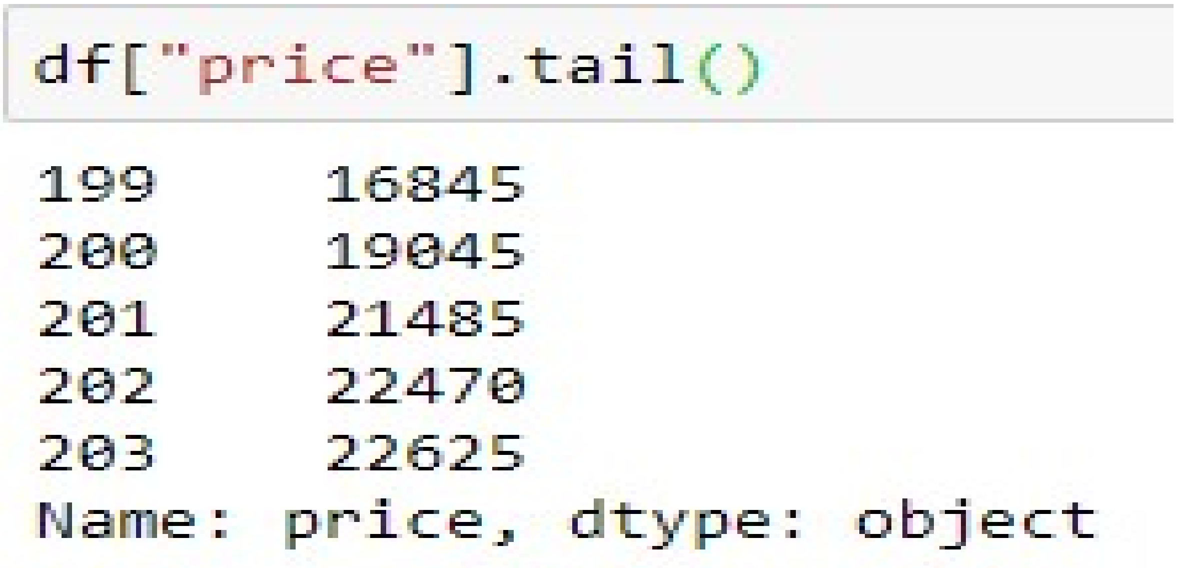
City
New York
New York
New York
New York

Formatted:

- more clear
- easy to aggregate
- easy to compare

Incorrect Data Types

- Sometimes wrong datatype is assigned to a column.

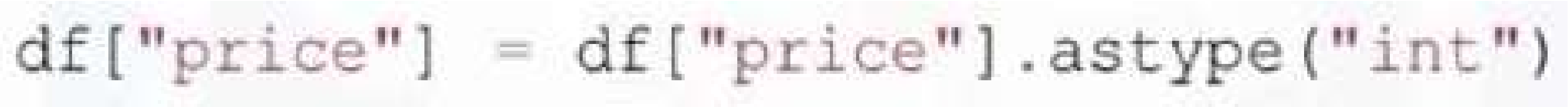


```
df["price"].tail()
```

```
199     16845
200     19045
201     21485
202     22470
203     22625
Name: price, dtype: object
```

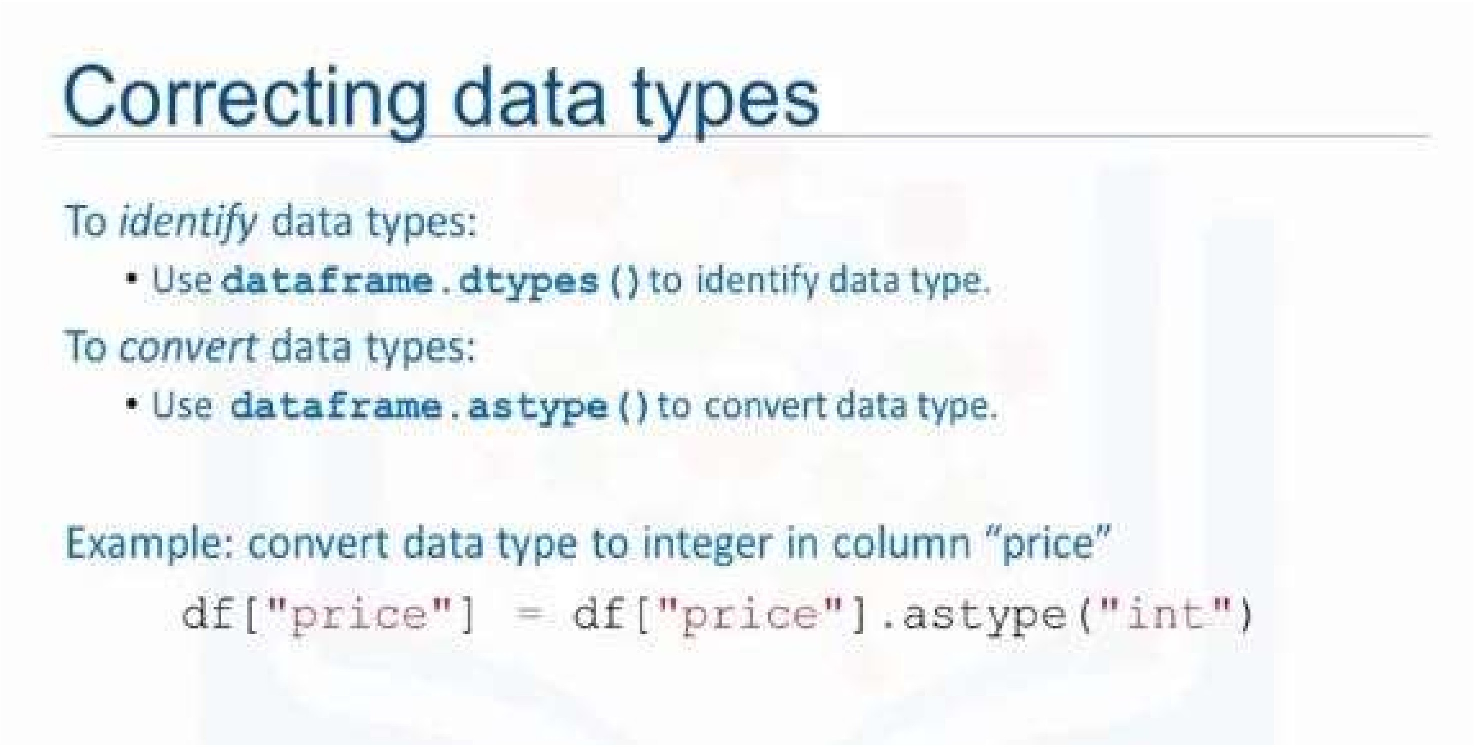
Correcting Data Types

- To identify data types:
  - Use dataframe.dtypes()
- To Convert data type:
  - Use dataframe.astype()



```
df["price"] = df["price"].astype("int")
```

Continue..



Correcting data types

To *identify* data types:

- Use `dataframe.dtypes()` to identify data type.

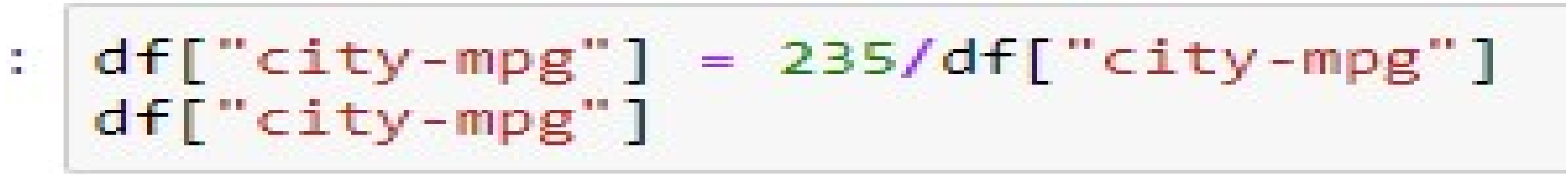
To *convert* data types:

- Use `dataframe.astype()` to convert data type.

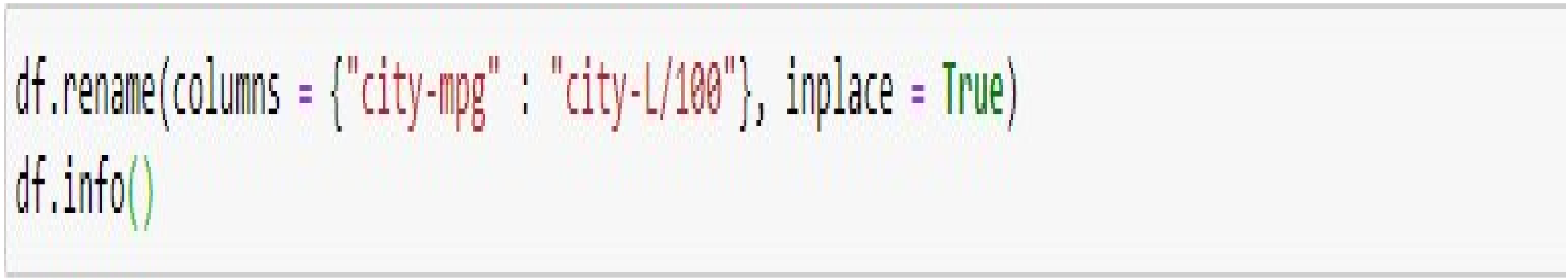
Example: convert data type to integer in column "price"

```
df["price"] = df["price"].astype("int")
```

Apply calculations to entire column



```
df["city-mpg"] = 235/df["city-mpg"]
df["city-mpg"]
```



```
df.rename(columns = {"city-mpg" : "city-L/100"}, inplace = True)
df.info()
```

Data Normalization

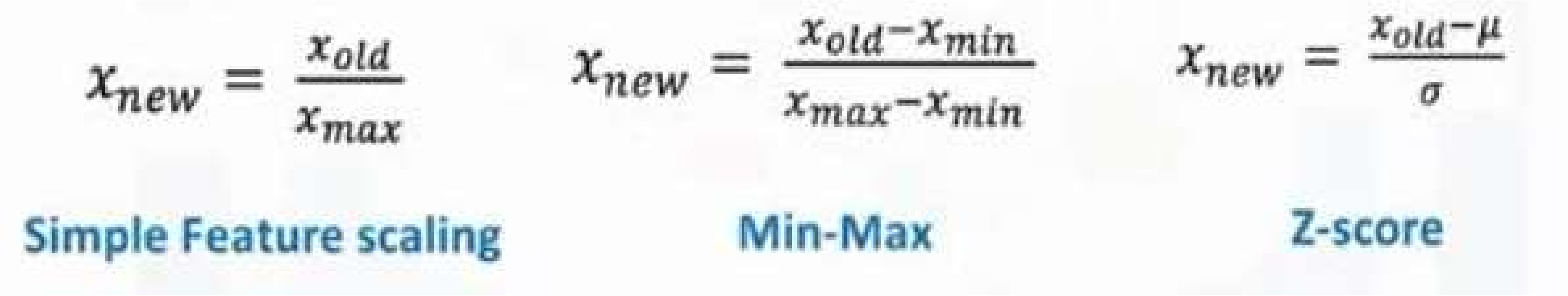
- Normalization is the process of transforming values of several variables into a similar range.
- Typical values range from 0 to 1

Normalization

Age	Income
20	20000
25	45000
37	28000

- Age and income are in different ranges..
- Hard to Compare.
- “Income” will influence the results more.

Methods for normalization



$$x_{new} = \frac{x_{old}}{x_{max}}$$

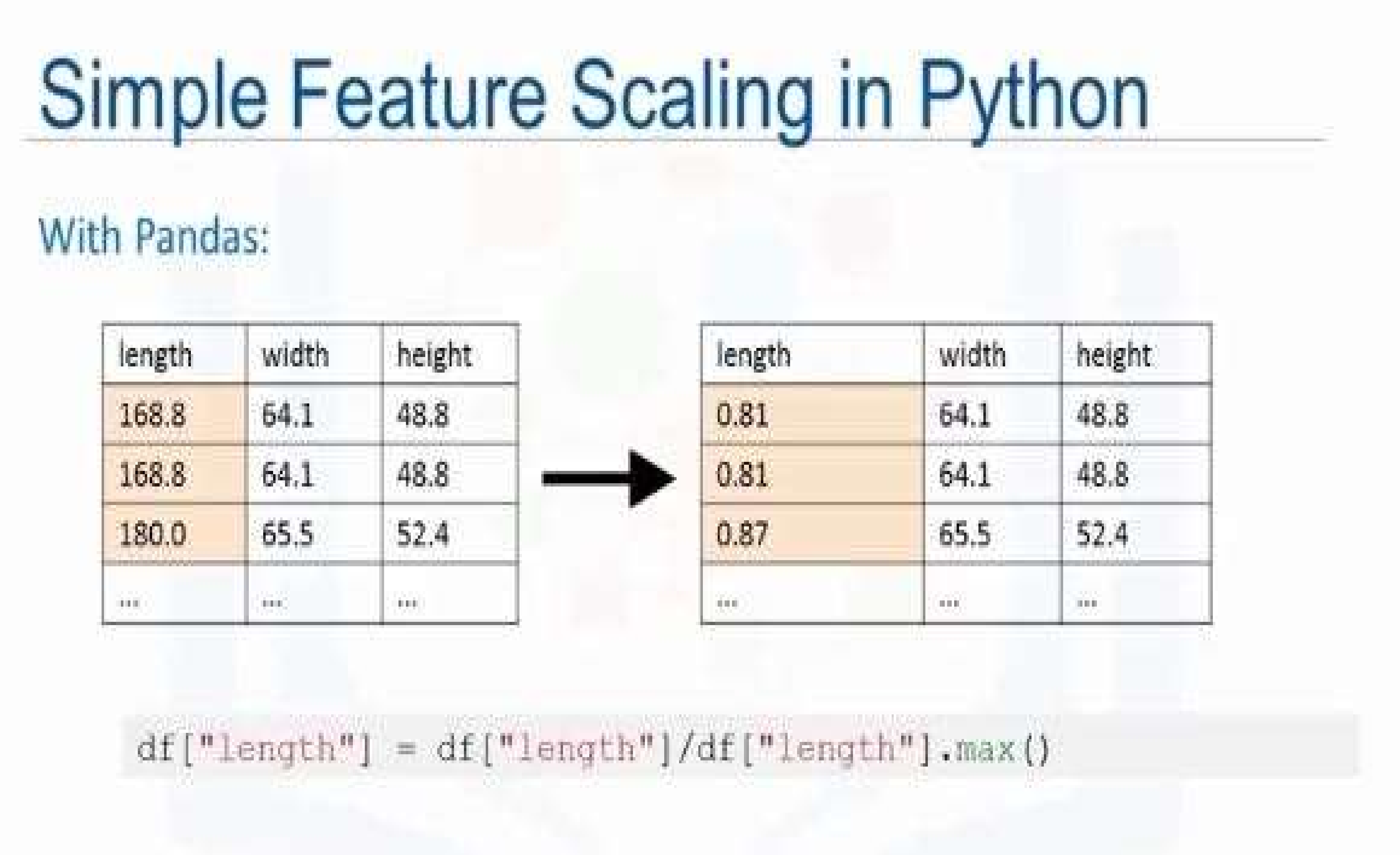
Simple Feature scaling

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

Min-Max

$$x_{new} = \frac{x_{old} - \mu}{\sigma}$$

Z-score



Simple Feature Scaling in Python

With Pandas:

length	width	height
168.8	64.1	48.8
168.8	64.1	48.8
180.0	65.5	52.4
...	...	...

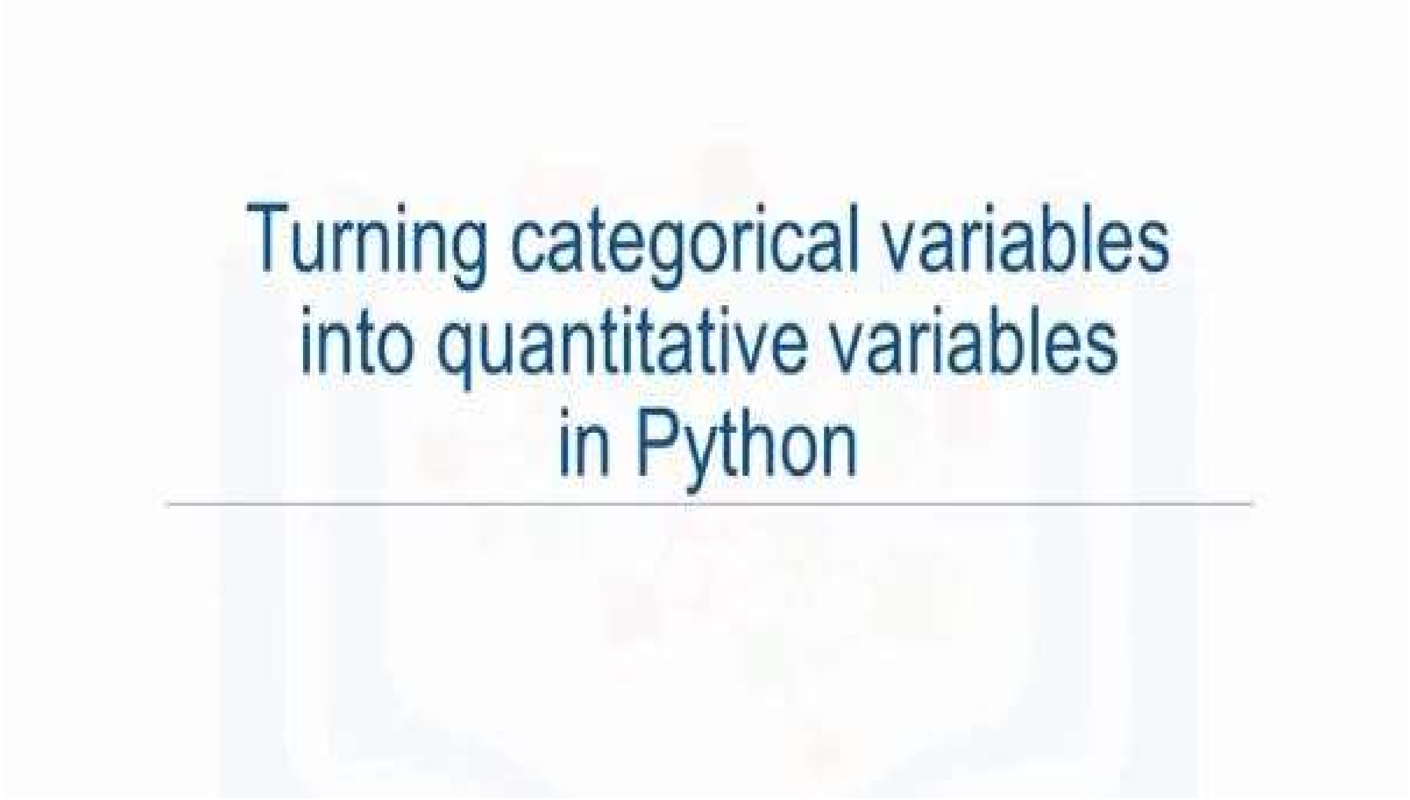
length	width	height
0.81	64.1	48.8
0.81	64.1	48.8
0.87	65.5	52.4
...	...	...

```
df["length"] = df["length"]/df["length"].max()
```

Simple feature scaling

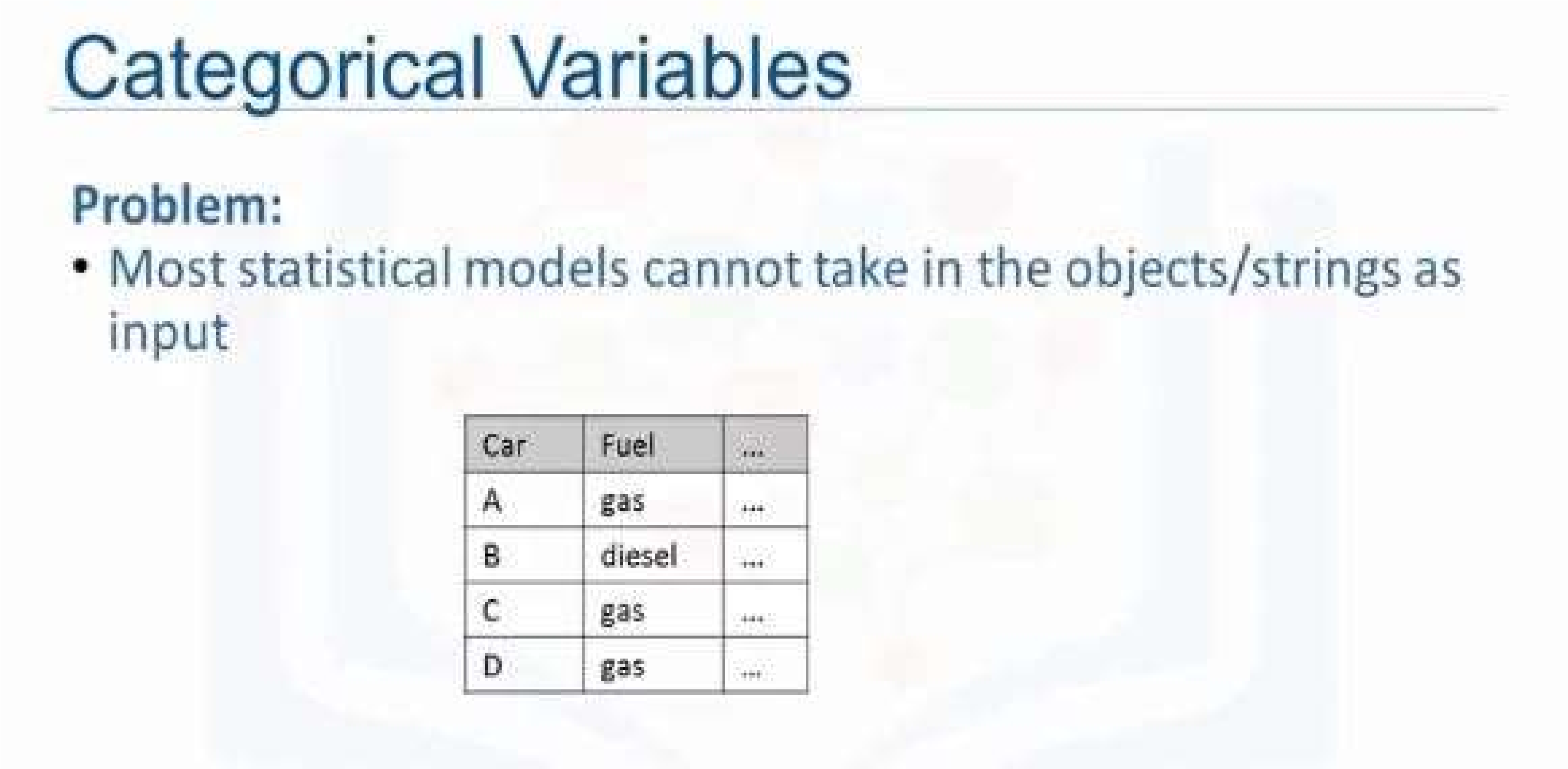
- df['length'] = df['length']/df['length'].max()
- df['width'] = df['width']/df['width'].max()

Categorical



Turning categorical variables into quantitative variables in Python

Continue..



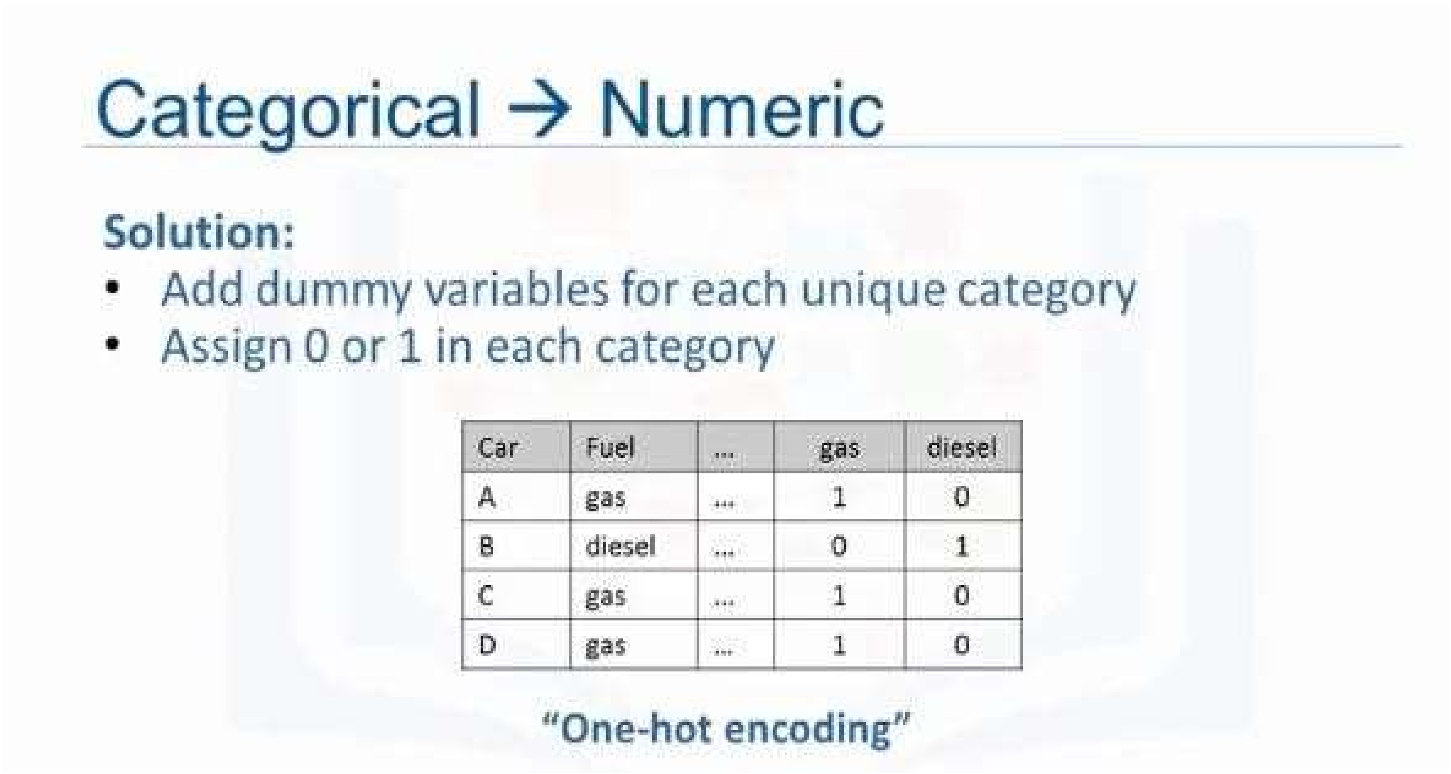
Categorical Variables

Problem:

- Most statistical models cannot take in the objects/strings as input

Car	Fuel	...
A	gas	...
B	diesel	...
C	gas	...
D	gas	...

Continue..



Categorical → Numeric

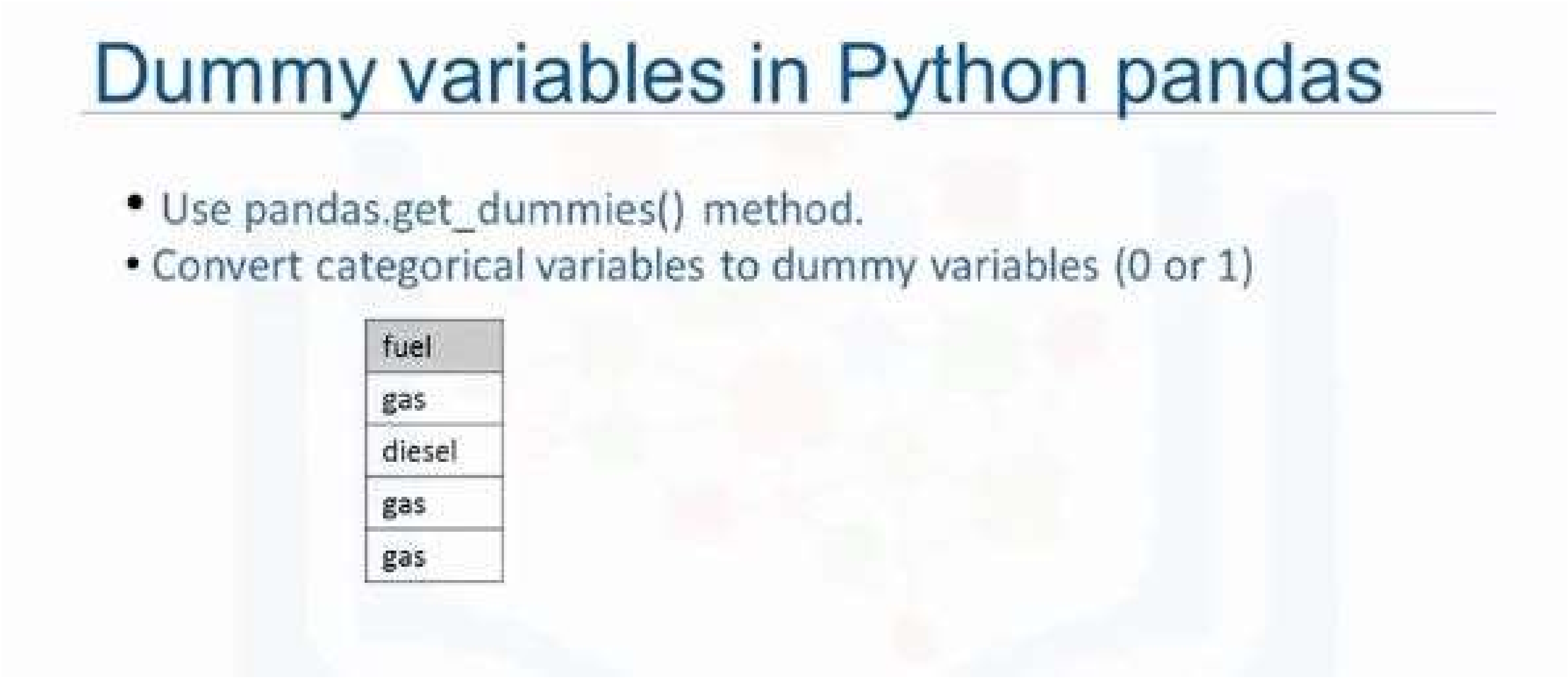
Solution:

- Add dummy variables for each unique category
- Assign 0 or 1 in each category

Car	Fuel	...	gas	diesel
A	gas	...	1	0
B	diesel	...	0	1
C	gas	...	1	0
D	gas	...	1	0

"One-hot encoding"

Continue..

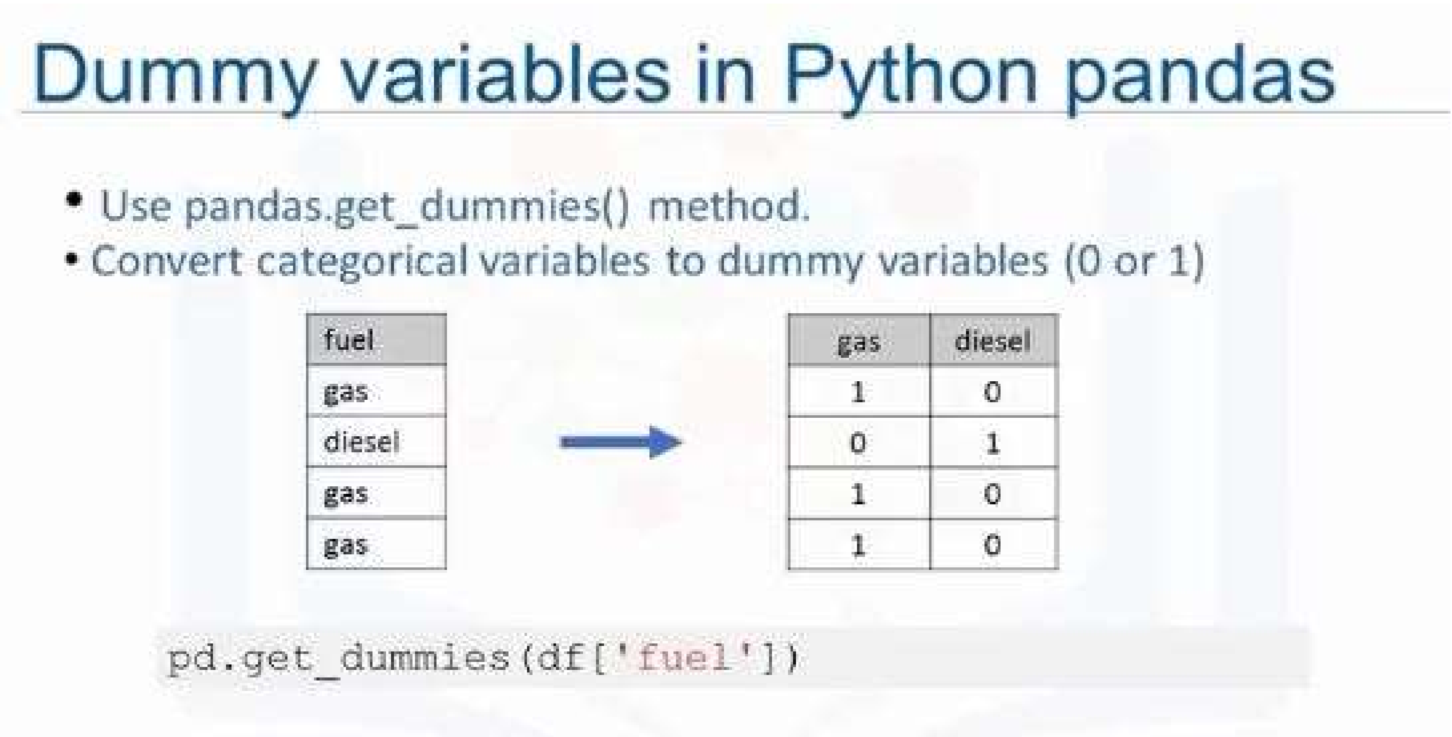


Dummy variables in Python pandas

- Use pandas.get_dummies() method.
- Convert categorical variables to dummy variables (0 or 1)

fuel
gas
diesel
gas
gas

Continue..



Dummy variables in Python pandas

- Use pandas.get_dummies() method.
- Convert categorical variables to dummy variables (0 or 1)

fuel
gas
diesel
gas
gas

→

gas	diesel
1	0
0	1
1	0
1	0

```
pd.get_dummies(df['fuel'])
```

Exploratory Data Analysis (EDA)

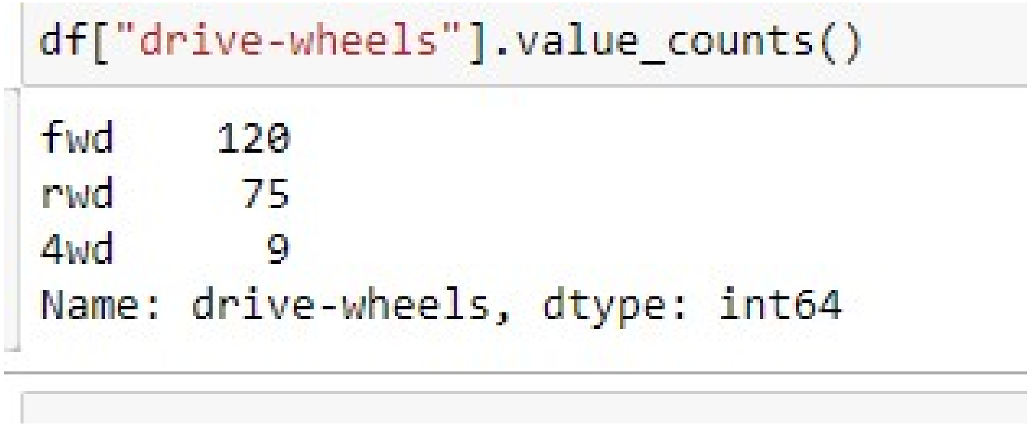
- Preliminary Step to data analysis
  - Get better understanding of data set.
  - Summarize main characteristics of data set.
  - Uncover relationship between different variables
  - Extract Important Variables

Descriptive Statistics

- Describe basic features of data.
- Giving short summaries about sample and measures of data set.

Descriptive Statistics

- df.describe()
- df.value_count()
  - Summarizing categorical data
  - Example : df[“drive-wheels"].value_counts()

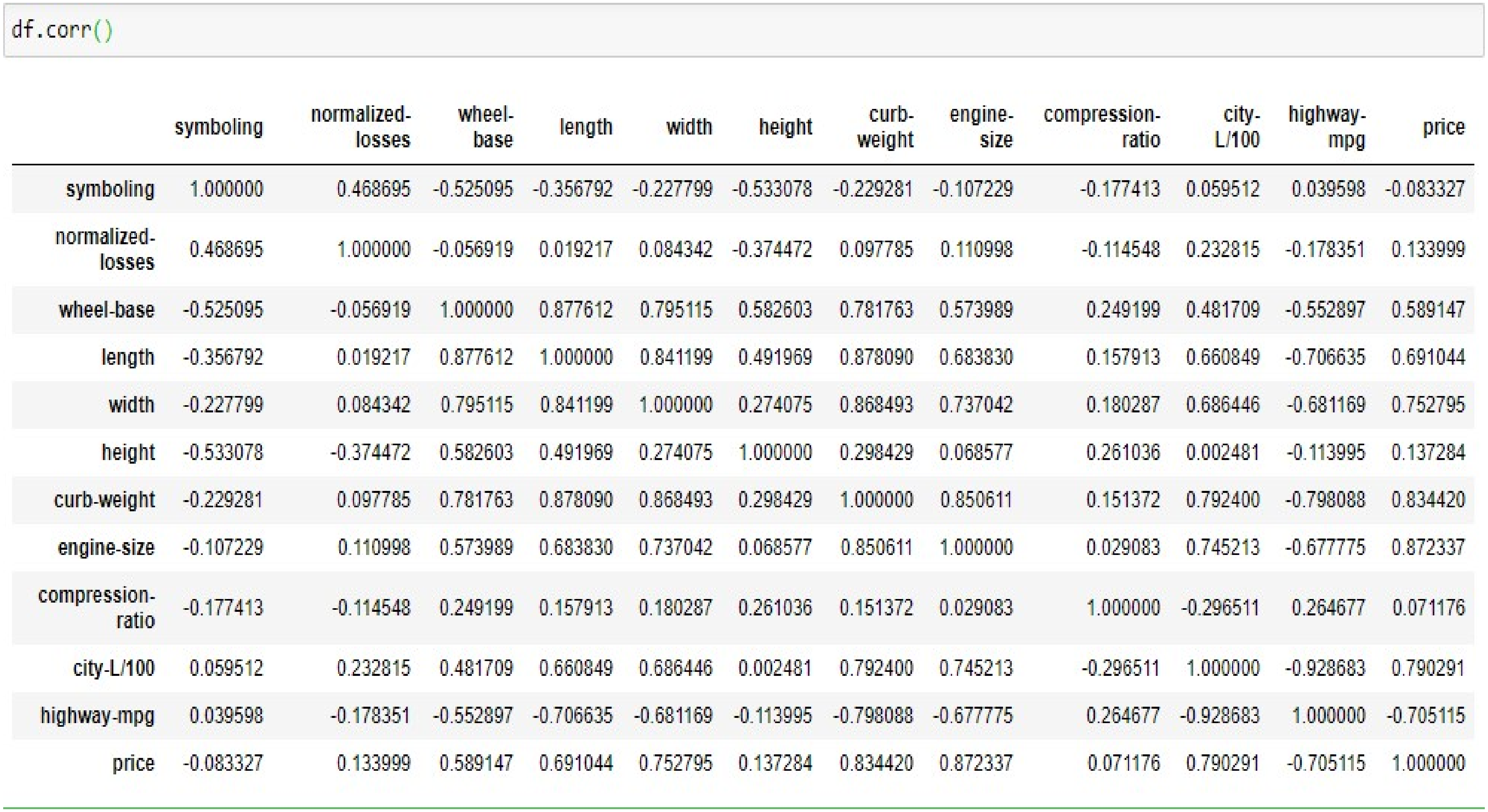


```
df["drive-wheels"].value_counts()
```

```
fwd    120
rwd     75
4wd      9
Name: drive-wheels, dtype: int64
```

Correlation

- a measure of the extent of interdependence between variables.
  - **1**: Total positive linear correlation.
  - **0**: No linear correlation, the two variables most likely do not affect each other.
  - **-1**: Total negative linear correlation.
- df.corr()



```
df.corr()
```

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-L/100	highway-mpg	price
symboling	1.000000	0.468695	-0.525095	-0.356792	-0.227799	-0.533078	-0.229281	-0.107229	-0.177413	0.059512	0.039598	-0.083327
normalized-losses	0.468695	1.000000	-0.056919	0.019217	0.084342	-0.374472	0.097785	0.110998	-0.114548	0.232815	-0.178351	0.133999
wheel-base	-0.525095	-0.056919	1.000000	0.877612	0.795115	0.582603	0.781763	0.573989	0.249199	0.481709	-0.552897	0.589147
length	-0.356792	0.019217	0.877612	1.000000	0.841199	0.491969	0.878090	0.683830	0.157913	0.660849	-0.706635	0.691044
width	-0.227799	0.084342	0.795115	0.841199	1.000000	0.274075	0.868493	0.737042	0.180287	0.686446	-0.681169	0.752795
height	-0.533078	-0.374472	0.582603	0.491969	0.274075	1.000000	0.298429	0.068577	0.261036	0.002481	-0.113995	0.137284
curb-weight	-0.229281	0.097785	0.781763	0.878090	0.868493	0.298429	1.000000	0.850611	0.151372	0.792400	-0.798088	0.834420
engine-size	-0.107229	0.110998	0.573989	0.683830	0.737042	0.068577	0.850611	1.000000	0.029083	0.745213	-0.677775	0.872337
compression-ratio	-0.177413	-0.114548	0.249199	0.157913	0.180287	0.261036	0.151372	0.029083	1.000000	-0.296511	0.264677	0.071176
city-L/100	0.059512	0.232815	0.481709	0.660849	0.686446	0.002481	0.792400	0.745213	-0.296511	1.000000	-0.928683	0.790291
highway-mpg	0.039598	-0.178351	-0.552897	-0.706635	-0.681169	-0.113995	-0.798088	-0.677775	0.264677	-0.928683	1.000000	-0.705115
price	-0.083327	0.133999	0.589147	0.691044	0.752795	0.137284	0.834420	0.872337	0.071176	0.790291	-0.705115	1.000000

Popular plotting libraries in Python

Python offers multiple graphing libraries that offers diverse features

• *matplotlib*	• to create 2D graphs and plots
• ***pandas visualization***	• easy to use interface, built on Matplotlib
• ***seaborn***	• provides a high-level interface for drawing attractive and informative statistical graphics
• ***ggplot***	• based on R's ggplot2, uses Grammar of Graphics
• ***plotly***	• can create interactive plots

Matplotlib

- 2D ploting library which produces good quality of figures.
- Although it has its origin in emulating the MATLAB graphics commands, it is independent of MATLAB.
- It makes heavy use of numpy and other extension code to provide good performance even for large arrays.

Scatter Plot

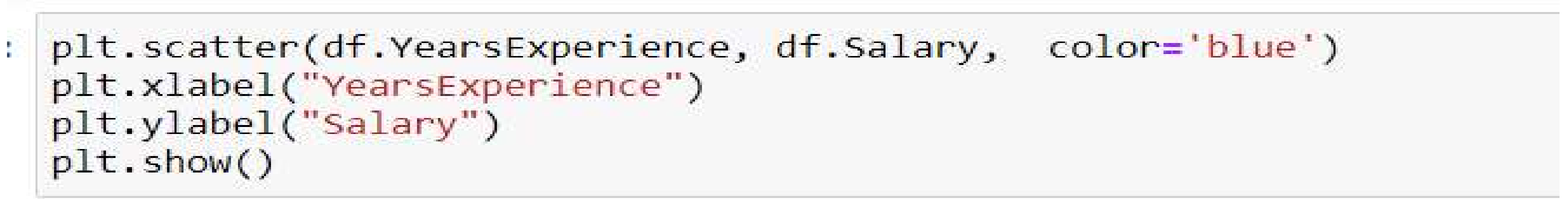
What is a scatter plot?

- A scatter plot is a set of points that represents the values obtained for two different variables plotted on a horizontal and vertical axes

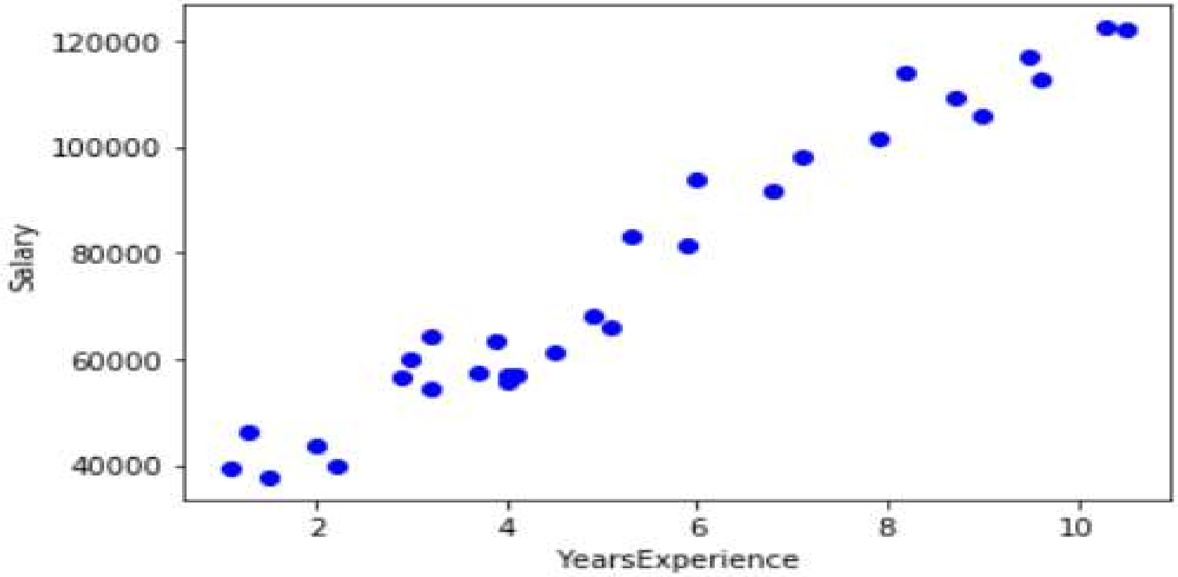
When to use scatter plots?

- Scatter plots are used to convey the relationship between two numerical variables
- Scatter plots are sometimes called correlation plots because they show how two variables are correlated

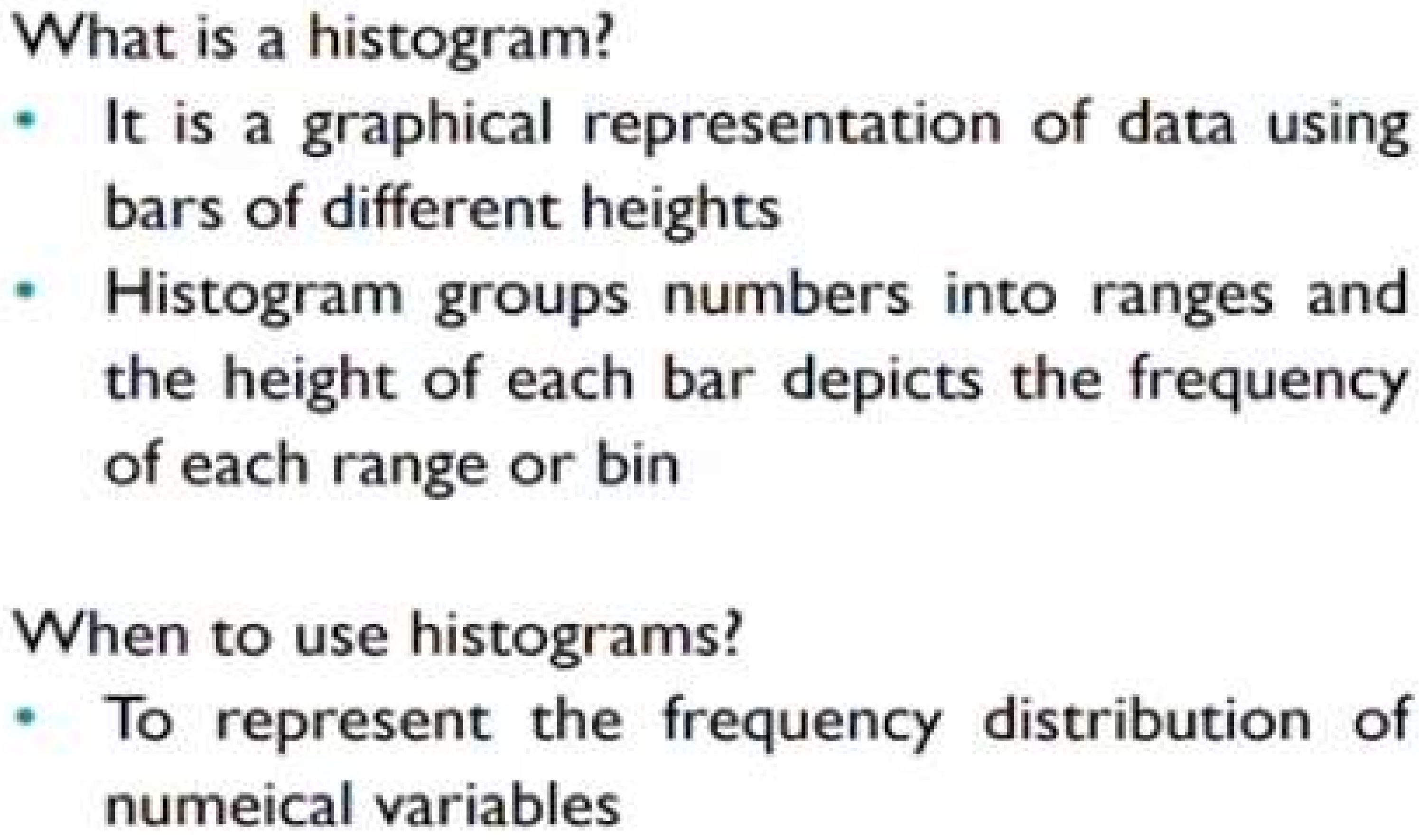
Plot Data



```
plt.scatter(df.YearsExperience, df.Salary,  color='blue')
plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.show()
```



Histogram



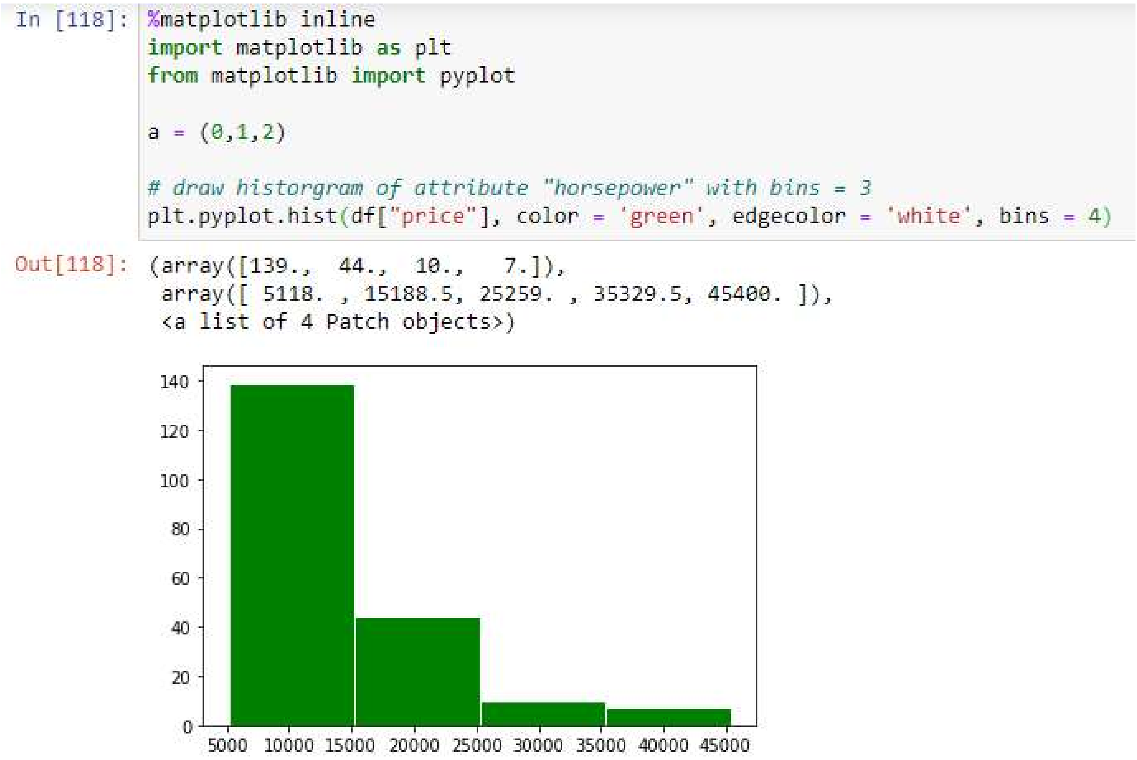
What is a histogram?

- It is a graphical representation of data using bars of different heights
- Histogram groups numbers into ranges and the height of each bar depicts the frequency of each range or bin

When to use histograms?

- To represent the frequency distribution of numeical variables

Histogram



```
In [118]: %matplotlib inline
          import matplotlib as plt
          from matplotlib import pyplot

          a = (0,1,2)

          # draw historgram of attribute "horsepower" with bins = 3
          plt.pyplot.hist(df["price"], color = 'green', edgecolor = 'white', bins = 4)
```

```
Out[118]: (array([139.,  44.,  10.,   7.]),
           array([ 5118. , 15188.5, 25259. , 35329.5, 45400. ]),
           <a list of 4 Patch objects>)
```

Seaborn

- Data visualization library based upon matplotlib.
- Provides high level interface for drawing attractive and informative statistical graphics.