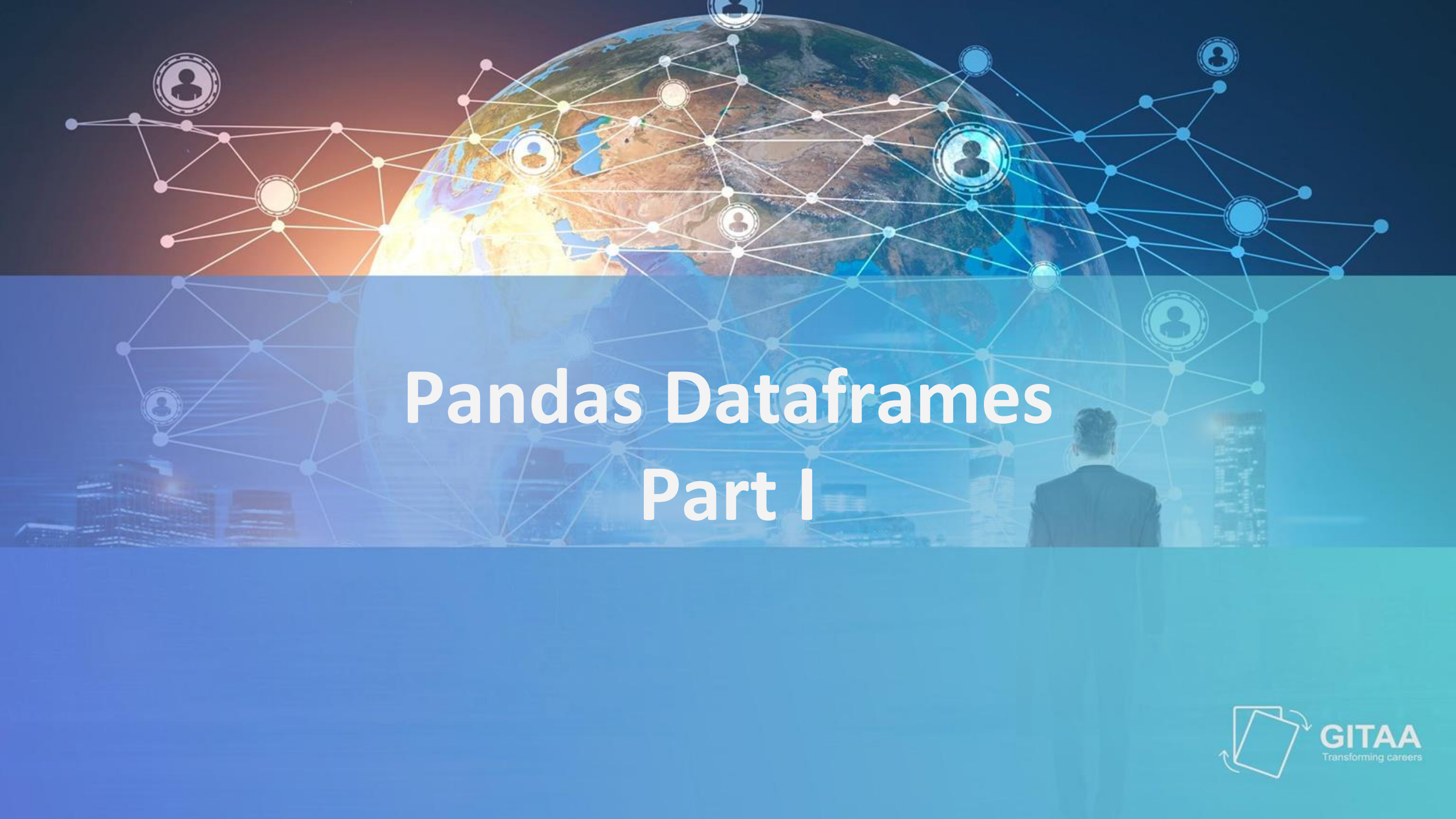


Pandas Dataframes
Part I

GITAA
Transforming careers

In this lecture

- Introduction to pandas
- Importing data into Spyder
- Creating copy of original data
- Attributes of data
- Indexing and selecting data

Introduction to Pandas

- Provides high-performance, easy-to-use data structures and analysis tools for the Python programming language
- Open-source Python library providing high-performance data manipulation and analysis tool using its powerful data structures
- Name pandas is derived from the word Panel Data – an econometrics term for multidimensional data

Pandas

- Pandas deals with dataframes

Name	Dimension	Description
Dataframe	2	• two-dimensional size-mutable • potentially heterogeneous tabular data structure with labeled axes (rows and columns)

Importing data into Spyder

- Importing necessary libraries

```
import os
```
← 'os' library to change the working directory

```
import pandas as pd
```
← 'pandas' library to work with dataframes

```
import numpy as np
```
← 'numpy' library to perform numeric operations

- Changing the working directory

```
os. chdir("D:\Pandas")
```

Importing data into Spyder

- Importing data

```
cars_data = pd.read_csv('Toyota.csv')
```

Index	Unnamed: 0	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	0	13500	23	46986	Diesel	90	1	0	2000	three	1165
1	1	13750	23	72937	Diesel	90	1	0	2000	3	1165
2	2	13950	24	41711	Diesel	90	nan	0	2000	3	1165

  - By passing `index_col=0` , first column becomes the index column

```
cars_data = pd.read_csv('Toyota.csv',index_col=0 )
```

Index	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	13500	23	46986	Diesel	90	1	0	2000	three	1165
1	13750	23	72937	Diesel	90	1	0	2000	3	1165
2	13950	24	41711	Diesel	90	nan	0	2000	3	1165

Creating copy of original data

- In Python, there are two ways to create copies
  - Shallow copy
  - Deep copy

	Shallow copy	*Deep copy*

Function	`samp=cars_data.copy(deep=False)`
	
	`samp = cars_data`
		`cars_data1=cars_data.copy(deep=True)`

Description	○ It only creates a new variable that shares the reference of the original object ○ Any changes made to a copy of object will be reflected in the original object as well	○ In case of deep copy, a copy of object is copied in other object with no reference to the original ○ Any changes made to a copy of object will not be reflected in the original object

Attributes of data

DataFrame.index

- **To get the index (row labels) of the dataframe**

```
cars_data1.index
```

```
Out[8]:
Int64Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
            ...
            1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435],
           dtype='int64', length=1436)
```

Attributes of data

DataFrame.columns

- **To get the column labels of the dataframe**

```
cars_data1.columns
```

```
Out[10]:
Index(['Price', 'Age', 'KM', 'FuelType',
'HP', 'MetColor', 'Automatic', 'CC',
       'Doors', 'Weight'],
      dtype='object')
```

Attributes of data

DataFrame.size

- **To get the total number of elements from the dataframe**

```
cars_data1.size
```

```
Out[11]: 14360
```

DataFrame.shape

- **To get the dimensionality of the dataframe**

```
cars_data1.shape
```

```
Out[12]: (1436, 10)
```
⟹ 1436 rows & 10 columns

Attributes of data

`DataFrame.memory_usage([index, deep])`

- **The memory usage of each column in bytes**

```
cars_data1.memory_usage()
```

⇒

```
Out[14]:
Index          11488
Price          11488
Age            11488
KM             11488
FuelType       11488
HP             11488
MetColor       11488
Automatic      11488
CC             11488
Doors          11488
Weight         11488
dtype: int64
```

`DataFrame.ndim`

- **The number of axes / array dimensions**

```
cars_data1.ndim
```

```
Out[15]: 2
```

A two-dimensional array stores data in a format consisting of rows and columns

Indexing and selecting data a

- Python slicing operator '[ ]' and attribute/ dot operator '.' are used for indexing

- Provides quick and easy access to pandas data structures

Indexing and selecting data

`DataFrame.head([n])`

- **The function head returns the first n rows from the dataframe**

```
cars_data1.head(6)
```

By default, the head() returns first 5 rows

```
Out[17]:
   Price   Age     KM FuelType  HP  MetColor  Automatic    CC  Doors  Weight
0  13500  23.0  46986   Diesel  90       1.0          0  2000  three    1165
1  13750  23.0  72937   Diesel  90       1.0          0  2000      3    1165
2  13950  24.0  41711   Diesel  90       NaN          0  2000      3    1165
3  14950  26.0  48000   Diesel  90       0.0          0  2000      3    1165
4  13750  30.0  38500   Diesel  90       0.0          0  2000      3    1170
5  12950  32.0  61000   Diesel  90       0.0          0  2000      3    1170
```

Indexing and selecting data

- **The function `tail` returns the last n rows for the object based on position**

```
cars_data1.tail(5)
```

```
Out[27]:
      Price   Age     KM FuelType   HP  MetColor  Automatic    CC Doors  Weight
1431   7500   NaN  20544   Petrol   86       1.0          0  1300     3    1025
1432  10845  72.0     ??   Petrol   86       0.0          0  1300     3    1015
1433   8500   NaN  17016   Petrol   86       0.0          0  1300     3    1015
1434   7250  70.0     ??      NaN   86       1.0          0  1300     3    1015
1435   6950  76.0      1   Petrol  110       0.0          0  1600     5    1114
```

- ✓ It is useful for quickly verifying data
- ✓ Ex: after sorting or appending rows.

Indexing and selecting data

- To access a scalar value, the fastest way is to use the at and iat methods
  - at provides label-based scalar lookups

```
In [29]: cars_data1.at[4,'FuelType']
Out[29]: 'Diesel'
```

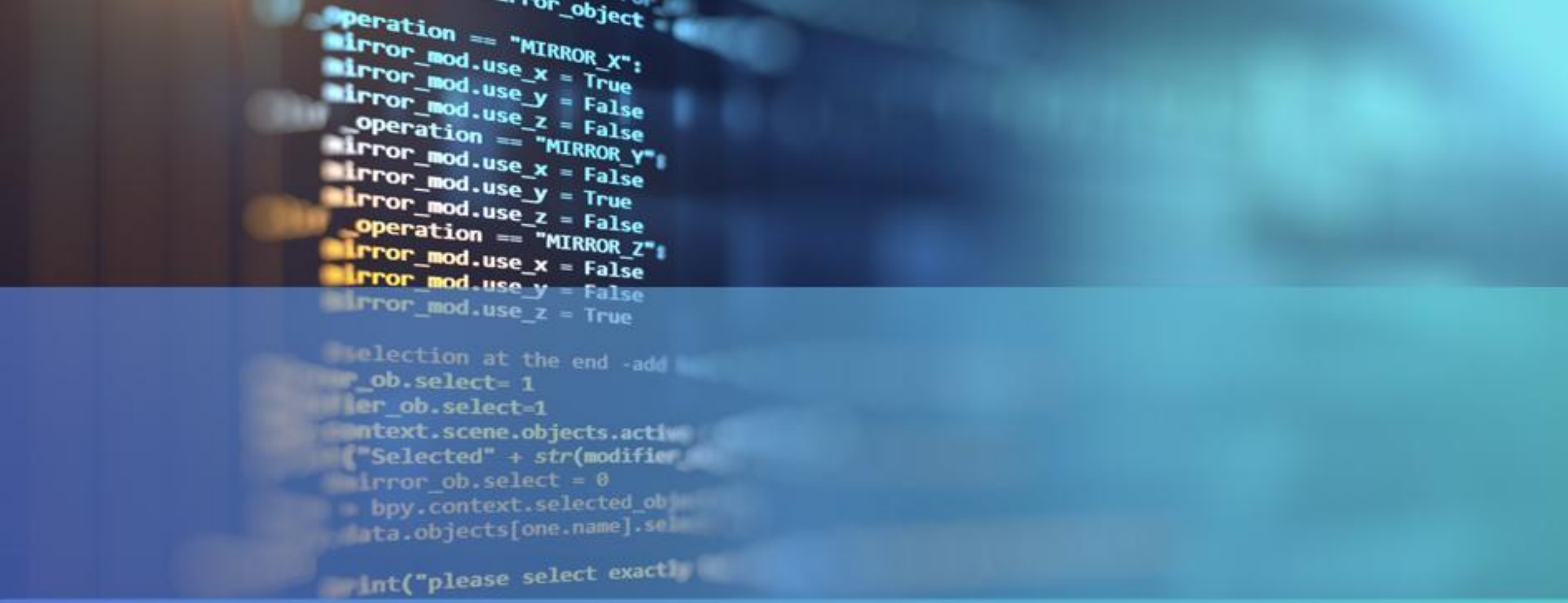
  - iat provides integer-based lookups

```
In [30]: cars_data1.iat[5,6]
Out[30]: 0
```

Indexing and selecting data

- To access a group of rows and columns by label(s) .loc[ ] can be used

```
In [31]: cars_data1.loc[:,'FuelType']
Out[31]:
0       Diesel
1       Diesel
2       Diesel
3       Diesel
4       Diesel
5       Diesel
6       Diesel
7          NaN
8       Petrol
```


operation == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True
selection at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.acti
("Selected" + str(modifier
irror_ob.select = 0
bpy.context.selected_ob
ata.objects[one.name].se
int("please select exact

THANK YOU