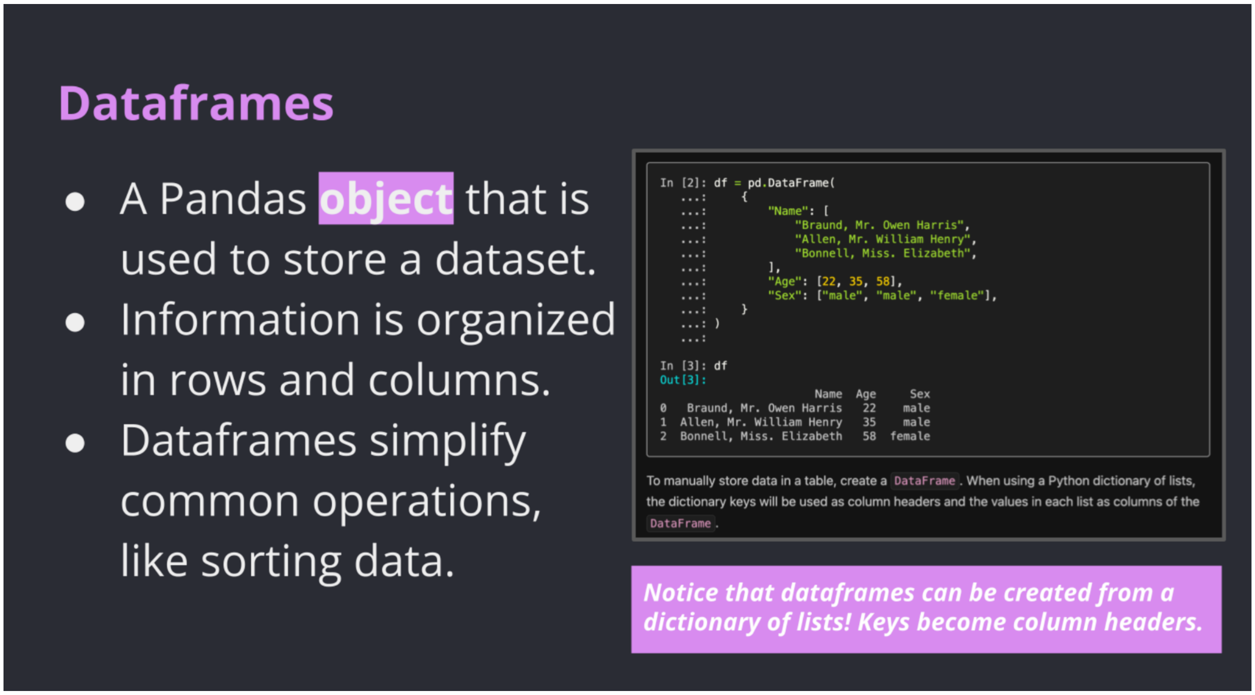
Pandas:

- A python library
- Makes analyzing data more efficient



Dataframes

- A Pandas **object** that is used to store a dataset.
- Information is organized in rows and columns.
- Dataframes simplify common operations, like sorting data.

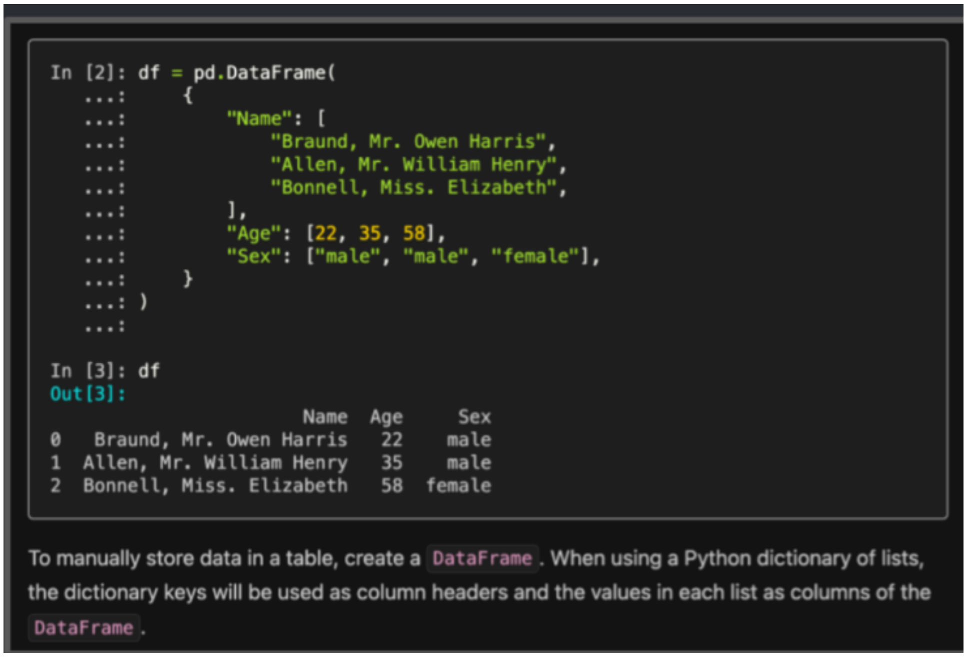
```
In [2]: df = pd.DataFrame(
   ...:     {
   ...:         "Name": [
   ...:             "Braund, Mr. Owen Harris",
   ...:             "Allen, Mr. William Henry",
   ...:             "Bonnell, Miss. Elizabeth",
   ...:         ],
   ...:         "Age": [22, 35, 58],
   ...:         "Sex": ["male", "male", "female"],
   ...:     }
   ...: )
   ...:

In [3]: df
Out[3]:
                       Name  Age     Sex
0   Braund, Mr. Owen Harris   22    male
1  Allen, Mr. William Henry   35    male
2  Bonnell, Miss. Elizabeth   58  female
```

To manually store data in a table, create a `DataFrame`. When using a Python dictionary of lists, the dictionary keys will be used as column headers and the values in each list as columns of the `DataFrame`.

Notice that dataframes can be created from a dictionary of lists! Keys become column headers.

Reading dictionaries into a dataframe:

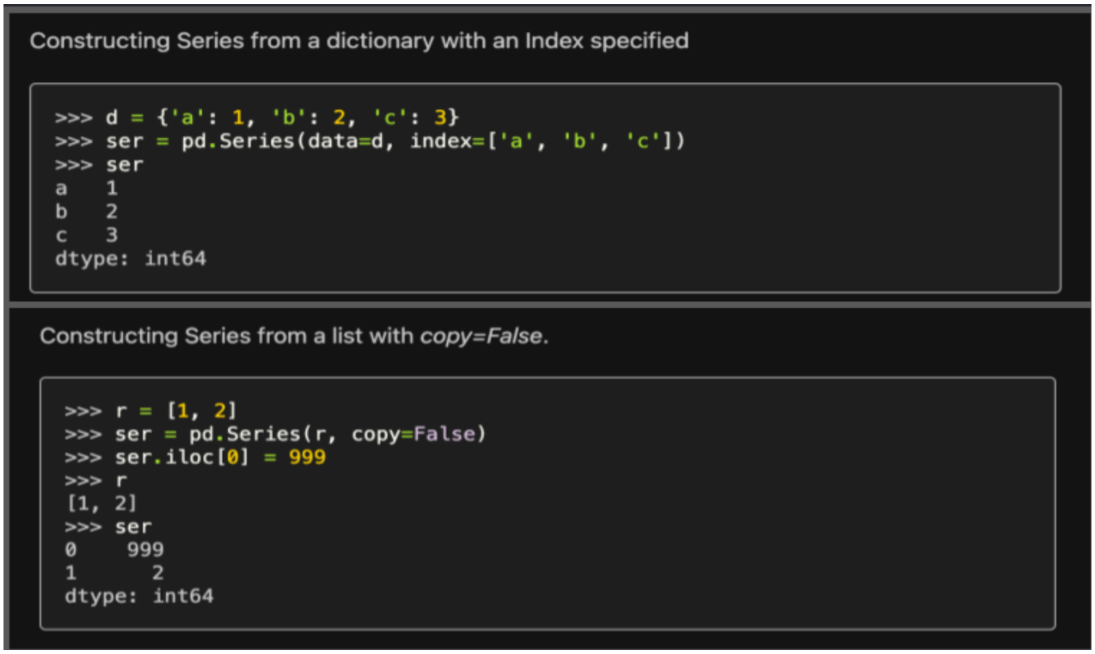


```
In [2]: df = pd.DataFrame(
   ...:     {
   ...:         "Name": [
   ...:             "Braund, Mr. Owen Harris",
   ...:             "Allen, Mr. William Henry",
   ...:             "Bonnell, Miss. Elizabeth",
   ...:         ],
   ...:         "Age": [22, 35, 58],
   ...:         "Sex": ["male", "male", "female"],
   ...:     }
   ...: )
   ...:

In [3]: df
Out[3]:
                       Name  Age     Sex
0   Braund, Mr. Owen Harris   22    male
1  Allen, Mr. William Henry   35    male
2  Bonnell, Miss. Elizabeth   58  female
```

To manually store data in a table, create a `DataFrame`. When using a Python dictionary of lists, the dictionary keys will be used as column headers and the values in each list as columns of the `DataFrame`.

Reading lists into a dataframe:



Constructing Series from a dictionary with an Index specified

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> ser = pd.Series(data=d, index=['a', 'b', 'c'])
>>> ser
a   1
b   2
c   3
dtype: int64
```

Constructing Series from a list with *copy=False*.

```
>>> r = [1, 2]
>>> ser = pd.Series(r, copy=False)
>>> ser.iloc[0] = 999
>>> r
[1, 2]
>>> ser
0    999
1      2
dtype: int64
```

Q1:

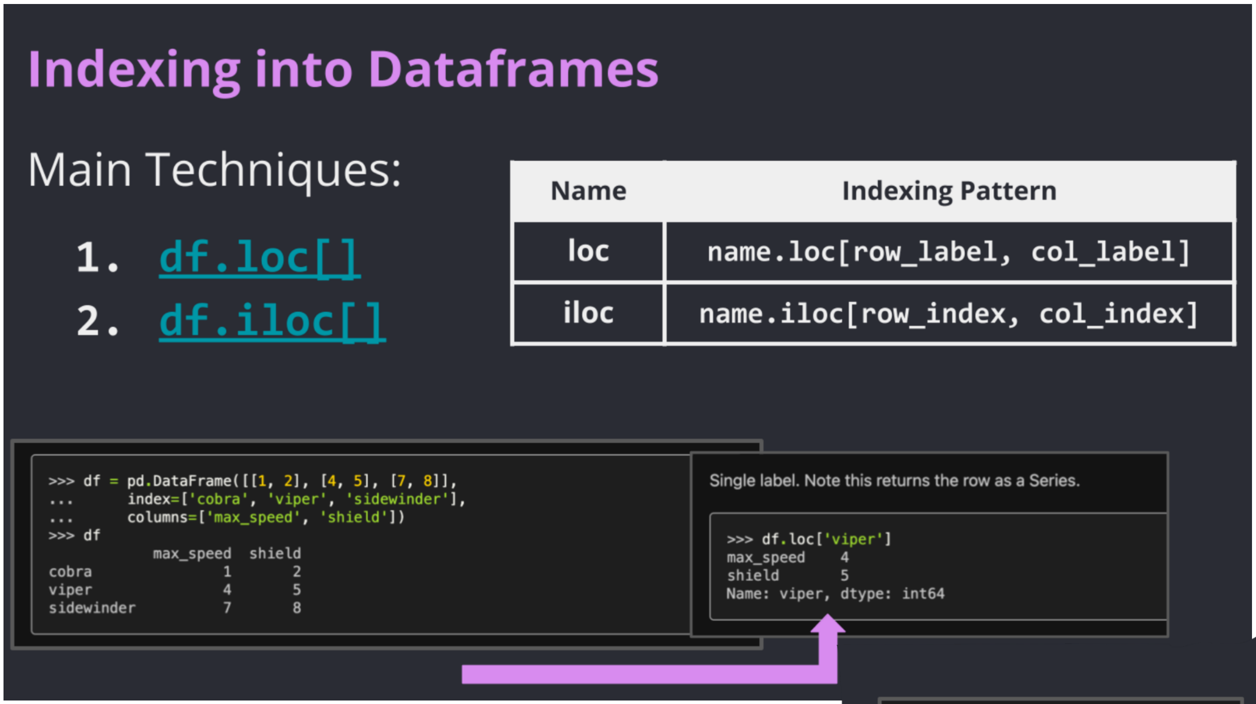
```
data =pd.DataFrame(
     {

     ‘A’=[1, 3, 5],
     ‘B’=[2, 4, 6]

     }

)
```

Notes continued below...



Indexing into Dataframes

Main Techniques:

1. df.loc[]
2. df.iloc[]

Name	Indexing Pattern
loc	name.loc[row_label, col_label]
iloc	name.iloc[row_index, col_index]

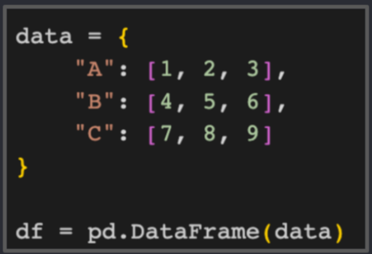
```
>>> df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
...      index=['cobra', 'viper', 'sidewinder'],
...      columns=['max_speed', 'shield'])
>>> df
            max_speed  shield
cobra               1       2
viper               4       5
sidewinder          7       8
```

Single label. Note this returns the row as a Series.

```
>>> df.loc['viper']
max_speed    4
shield       5
Name: viper, dtype: int64
```

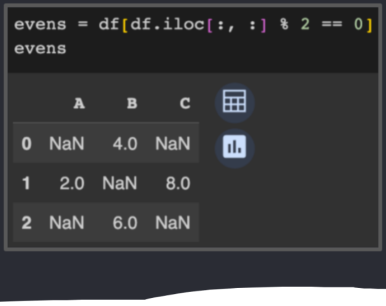
Q2:

```
midpoint = baseball.shape[0]//2
baseball.iloc[midpoint:]
```



```
data = {
    "A": [1, 2, 3],
    "B": [4, 5, 6],
    "C": [7, 8, 9]
}

df = pd.DataFrame(data)
```



```
evens = df[df.iloc[:, :] % 2 == 0]
evens
```

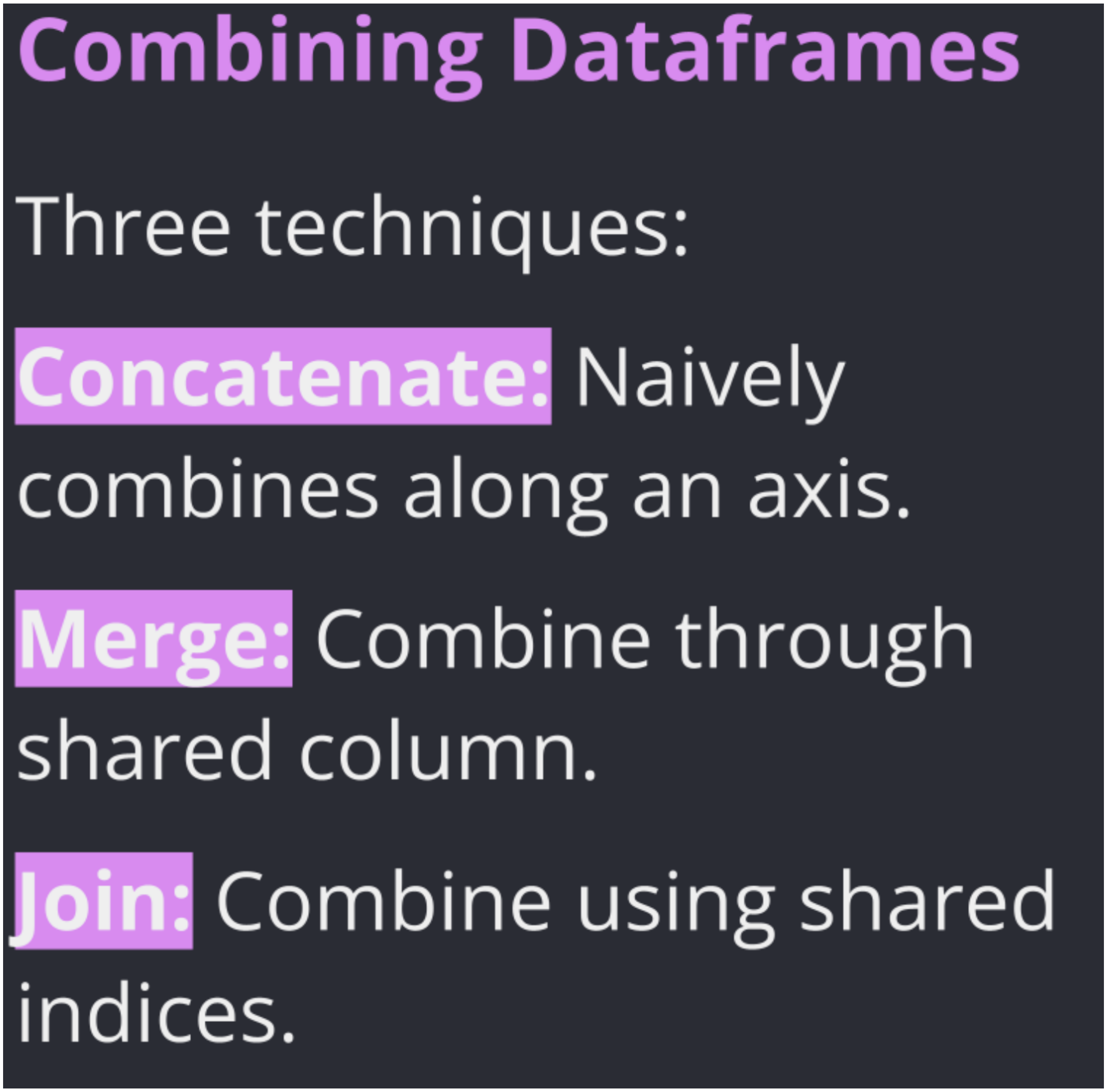
```
     A    B    C
0  NaN  4.0  NaN
1  2.0  NaN  8.0
2  NaN  6.0  NaN
```

Popular Pattern:

df[condition]

Q3:

```
df[df.loc[:, “Smoker”] == True]
```



Combining Dataframes

Three techniques:

Concatenate: Naively combines along an axis.

Merge: Combine through shared column.

Join: Combine using shared indices.