Introduction to spreadsheets

STREAMLINED DATA INGESTION WITH PANDAS

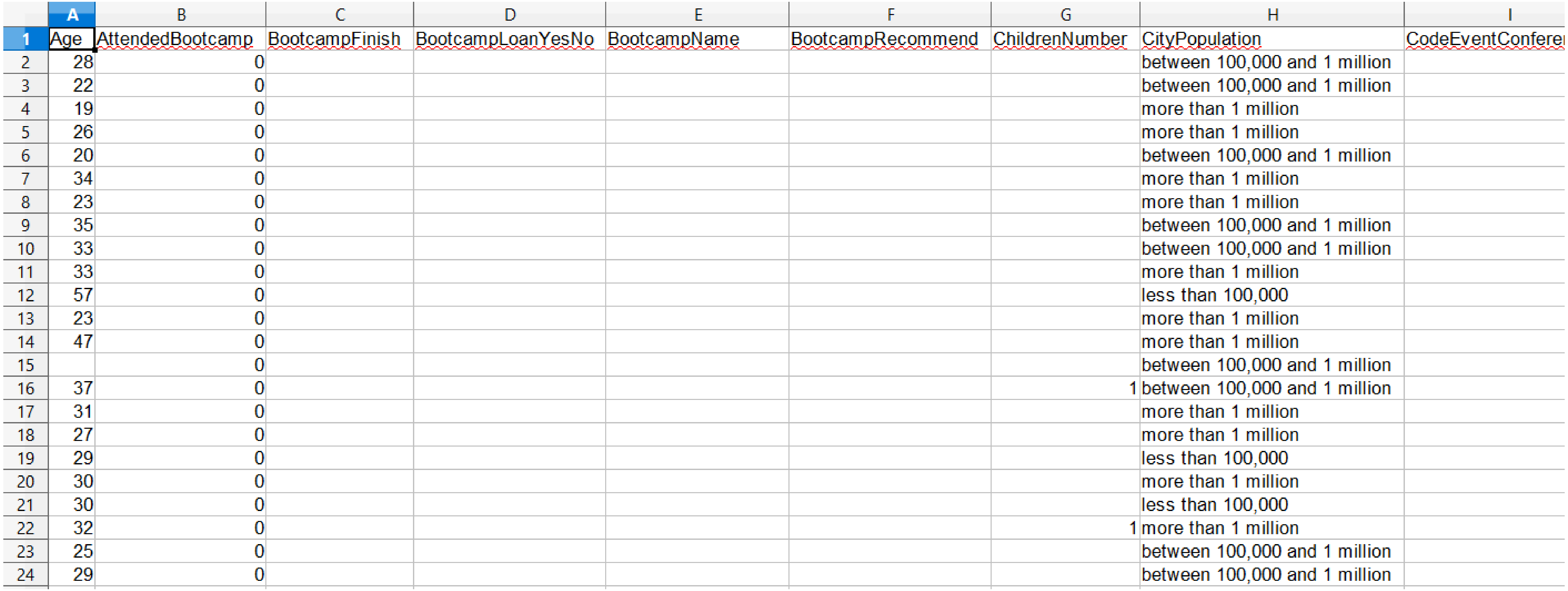
Amany Mahfouz
Instructor

Spreadsheets

- Also known as Excel files
- Data stored in tabular form, with cells arranged in rows and columns
- Unlike flat files, can have formatting and formulas
- Multiple spreadsheets can exist in a workbook

Loading Spreadsheets

- Spreadsheets have their own loading function in `pandas` : `read_excel()`



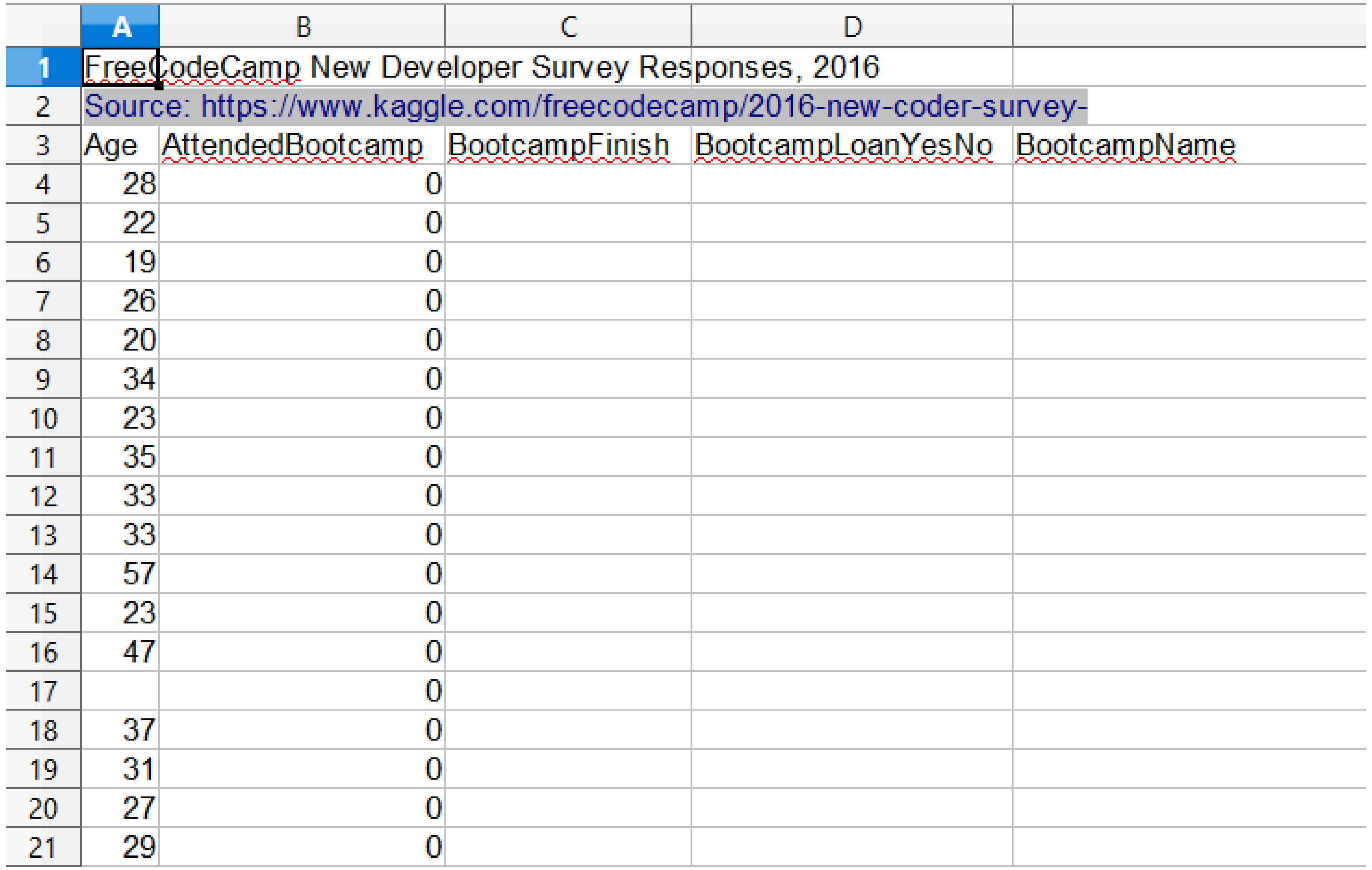
	A	B	C	D	E	F	G	H	I
1	Age	AttendedBootcamp	BootcampFinish	BootcampLoanYesNo	BootcampName	BootcampRecommend	ChildrenNumber	CityPopulation	CodeEventConfere
2	28	0						between 100,000 and 1 million	
3	22	0						between 100,000 and 1 million	
4	19	0						more than 1 million	
5	26	0						more than 1 million	
6	20	0						between 100,000 and 1 million	
7	34	0						more than 1 million	
8	23	0						more than 1 million	
9	35	0						between 100,000 and 1 million	
10	33	0						between 100,000 and 1 million	
11	33	0						more than 1 million	
12	57	0						less than 100,000	
13	23	0						more than 1 million	
14	47	0						more than 1 million	
15		0						between 100,000 and 1 million	
16	37	0					1	between 100,000 and 1 million	
17	31	0						more than 1 million	
18	27	0						more than 1 million	
19	29	0						less than 100,000	
20	30	0						more than 1 million	
21	30	0						less than 100,000	
22	32	0					1	more than 1 million	
23	25	0						between 100,000 and 1 million	
24	29	0						between 100,000 and 1 million	

Loading Spreadsheets

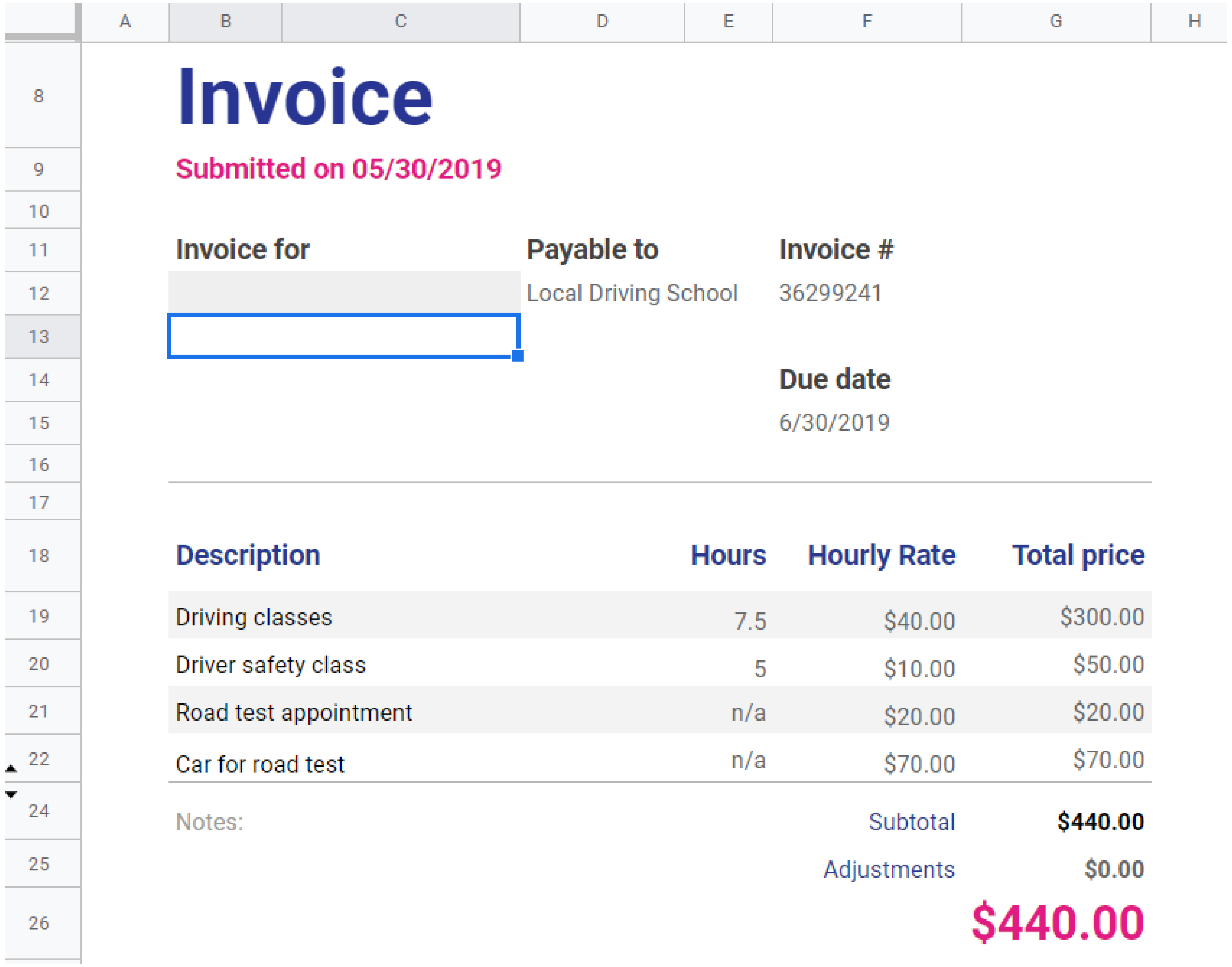
```
import pandas as pd

# Read the Excel file
survey_data = pd.read_excel("fcc_survey.xlsx")


# View the first 5 lines of data
print(survey_data.head())
```

```
    Age  AttendedBootcamp       ...                    SchoolMajor  StudentDebtOwe
0  28.0               0.0       ...                            NaN           20000
1  22.0               0.0       ...                            NaN             NaN
2  19.0               0.0       ...                            NaN             NaN
3  26.0               0.0       ...        Cinematography And Film            7000
4  20.0               0.0       ...                            NaN             NaN

[5 rows x 98 columns]
```

Loading Select Columns and Rows



	A	B	C	D	
1	FreeCodeCamp New Developer Survey Responses, 2016				
2	Source: https://www.kaggle.com/freecodecamp/2016-new-coder-survey-				
3	Age	AttendedBootcamp	BootcampFinish	BootcampLoanYesNo	BootcampName
4	28	0			
5	22	0			
6	19	0			
7	26	0			
8	20	0			
9	34	0			
10	23	0			
11	35	0			
12	33	0			
13	33	0			
14	57	0			
15	23	0			
16	47	0			
17		0			
18	37	0			
19	31	0			
20	27	0			
21	29	0			



Invoice

Submitted on 05/30/2019

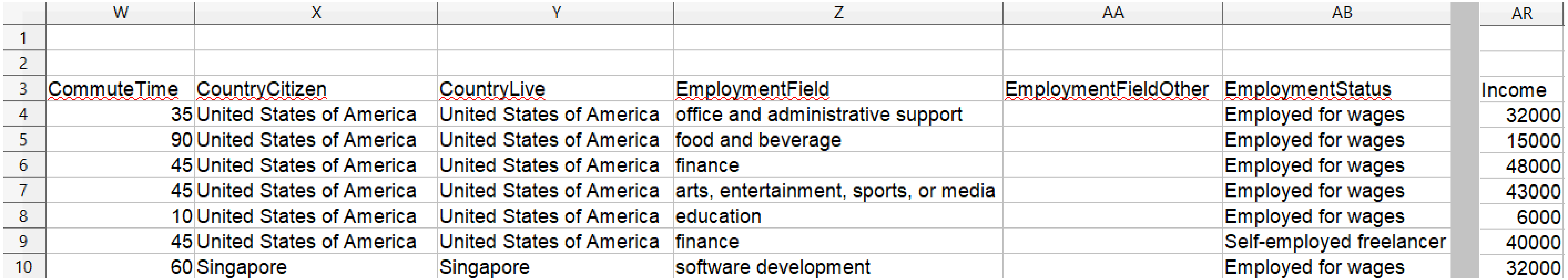
Invoice for	Payable to	Invoice #
	Local Driving School	36299241
		Due date
		6/30/2019

Description	Hours	Hourly Rate	Total price
Driving classes	7.5	$40.00	$300.00
Driver safety class	5	$10.00	$50.00
Road test appointment	n/a	$20.00	$20.00
Car for road test	n/a	$70.00	$70.00
Notes:		Subtotal	$440.00
		Adjustments	$0.00
			$440.00

Loading Select Columns and Rows

- `read_excel()` has many keyword arguments in common with `read_csv()`
  - `nrows` : limit number of rows to load
  - `skiprows` : specify number of rows or row numbers to skip
  - `usecols` : choose columns by name, positional number, or letter (e.g. "A:P")

Loading Select Columns and Rows



	W	X	Y	Z	AA	AB	AR
1							
2							
3	CommuteTime	CountryCitizen	CountryLive	EmploymentField	EmploymentFieldOther	EmploymentStatus	Income
4	35	United States of America	United States of America	office and administrative support		Employed for wages	32000
5	90	United States of America	United States of America	food and beverage		Employed for wages	15000
6	45	United States of America	United States of America	finance		Employed for wages	48000
7	45	United States of America	United States of America	arts, entertainment, sports, or media		Employed for wages	43000
8	10	United States of America	United States of America	education		Employed for wages	6000
9	45	United States of America	United States of America	finance		Self-employed freelancer	40000
10	60	Singapore	Singapore	software development		Employed for wages	32000

Loading Select Columns and Rows

```
# Read columns W-AB and AR of file, skipping metadata header
survey_data = pd.read_excel("fcc_survey_with_headers.xlsx",
                            skiprows=2,
                            usecols="W:AB, AR")


# View data
print(survey_data.head())
```

```
   CommuteTime            CountryCitizen  ...    EmploymentFieldOther    EmploymentStatus   Income
0         35.0  United States of America  ...                     NaN  Employed for wages  32000.0
1         90.0  United States of America  ...                     NaN  Employed for wages  15000.0
2         45.0  United States of America  ...                     NaN  Employed for wages  48000.0
3         45.0  United States of America  ...                     NaN  Employed for wages  43000.0
4         10.0  United States of America  ...                     NaN  Employed for wages   6000.0

[5 rows x 7 columns]
```

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS

Getting data from multiple worksheets

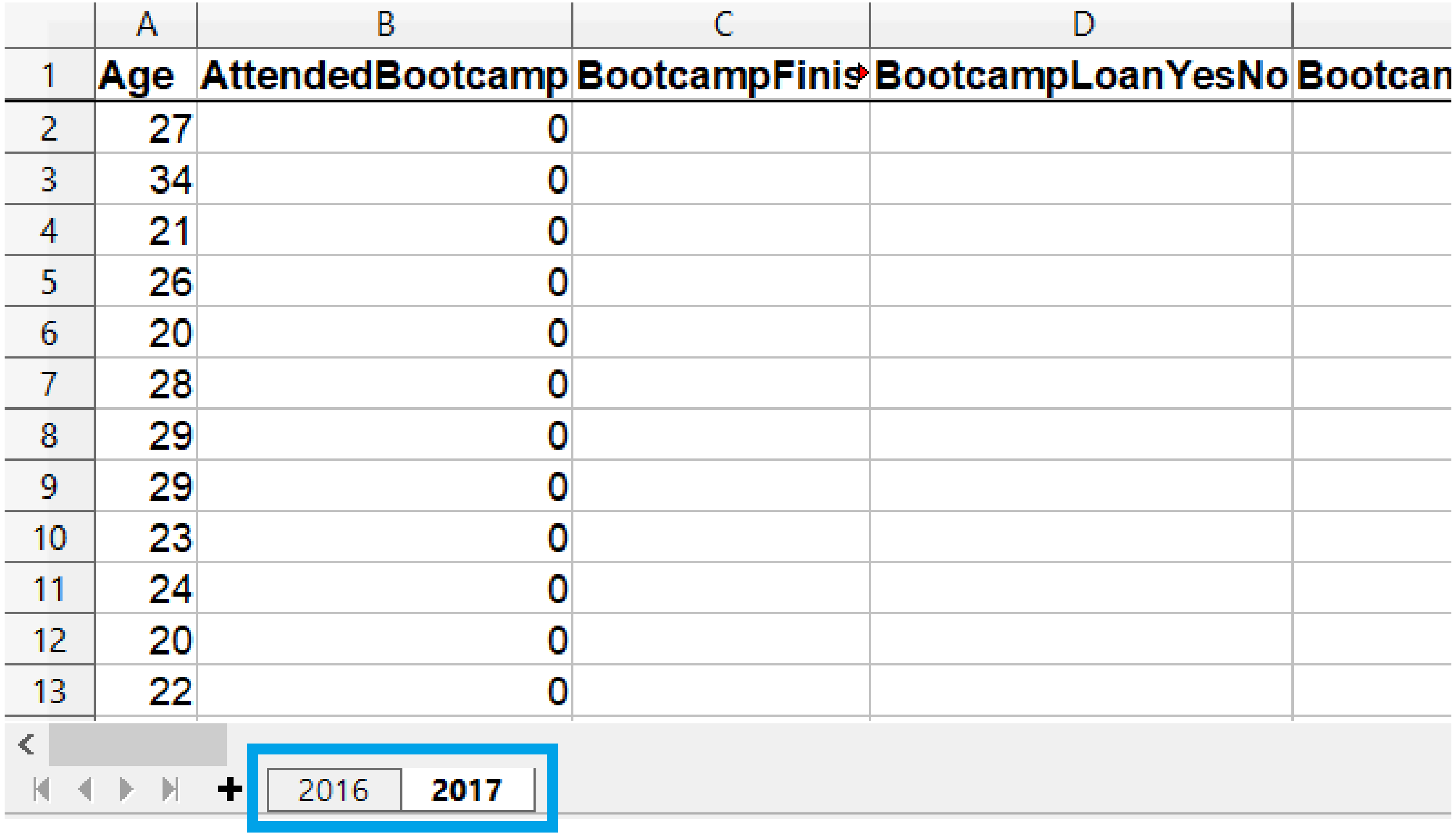
STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

Selecting Sheets to Load

- `read_excel()` loads the first sheet in an Excel file by default
- Use the `sheet_name` keyword argument to load other sheets
- Specify spreadsheets by name and/or (zero-indexed) position number
- Pass a list of names/numbers to load more than one sheet at a time
- Any arguments passed to `read_excel()` apply to all sheets read

Selecting Sheets to Load



	A	B	C	D	
1	Age	AttendedBootcamp	BootcampFinis	BootcampLoanYesNo	Bootcan
2	27	0			
3	34	0			
4	21	0			
5	26	0			
6	20	0			
7	28	0			
8	29	0			
9	29	0			
10	23	0			
11	24	0			
12	20	0			
13	22	0			

Loading Select Sheets

```
# Get the second sheet by position index
survey_data_sheet2 = pd.read_excel('fcc_survey.xlsx',
                                   sheet_name=1)


# Get the second sheet by name
survey_data_2017 = pd.read_excel('fcc_survey.xlsx',
                                 sheet_name='2017')


print(survey_data_sheet2.equals(survey_data_2017))
```

```
True
```

Loading All Sheets

- Passing `sheet_name=None` to `read_excel()` reads all sheets in a workbook

```
survey_data = pd.read_excel("fcc_survey.xlsx", sheet_name=None)

print(type(survey_data))
```

```
<class 'collections.OrderedDict'>
```

```
for key, value in survey_data.items():
    print(key, type(value))
```

```
2016 <class 'pandas.core.frame.DataFrame'>
2017 <class 'pandas.core.frame.DataFrame'>
```

Putting It All Together

```
# Create empty data frame to hold all loaded sheets
all_responses = pd.DataFrame()


# Iterate through data frames in dictionary
for sheet_name, frame in survey_responses.items():
    # Add a column so we know which year data is from
    frame["Year"] = sheet_name


    # Add each data frame to all_responses
    all_responses = all_responses.append(frame)


# View years in data
print(all_responses.Year.unique())
```

```
['2016' '2017']
```

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS

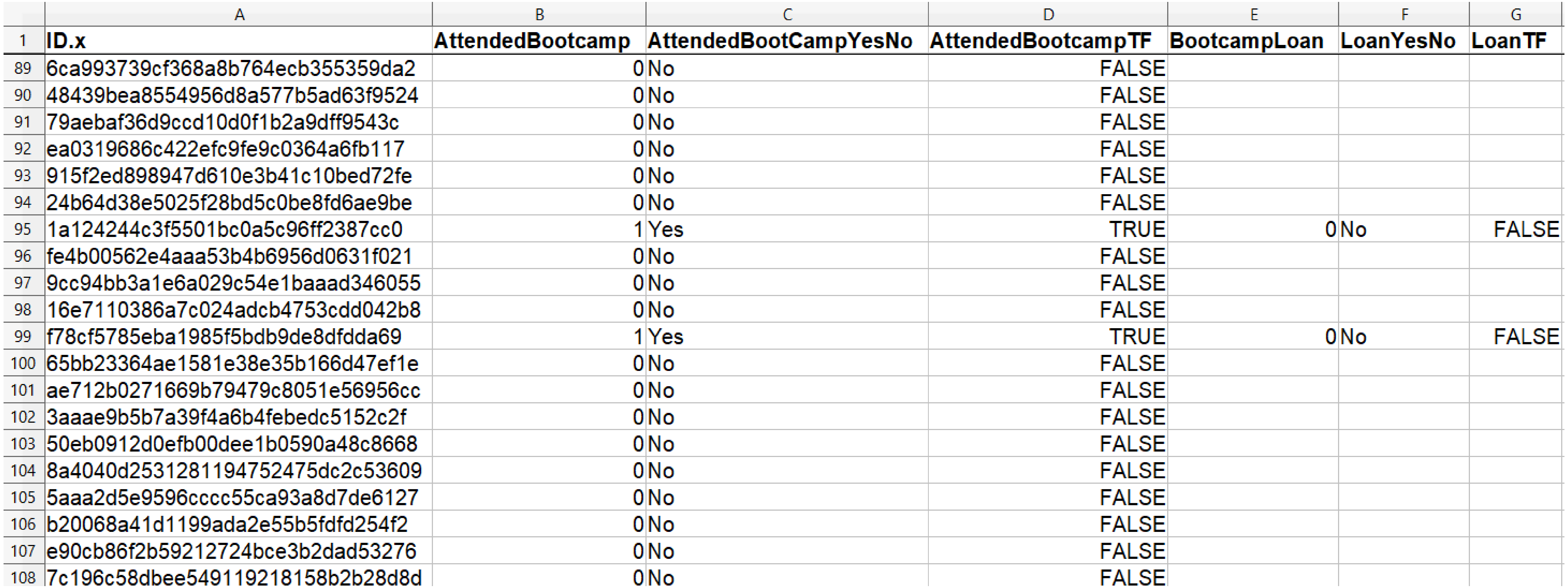
Modifying imports: true/false data

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

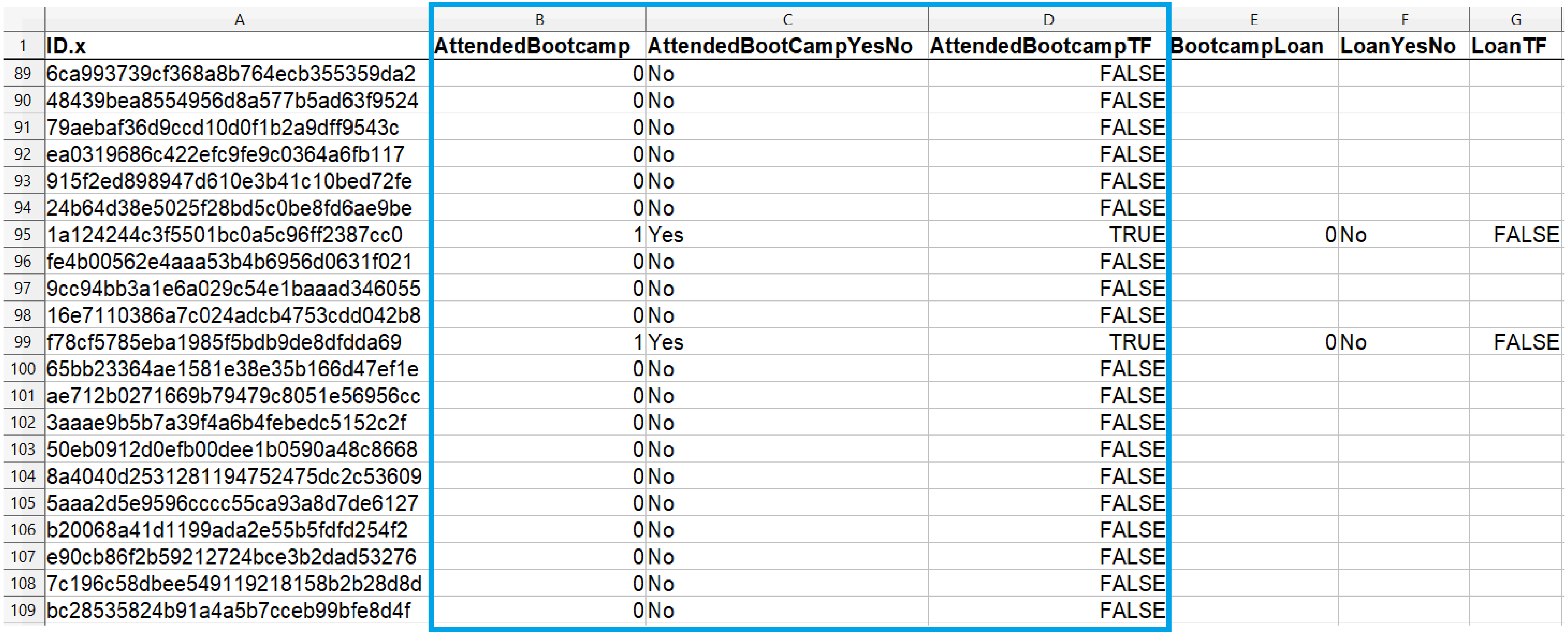
Boolean Data

- `True` / `False` data



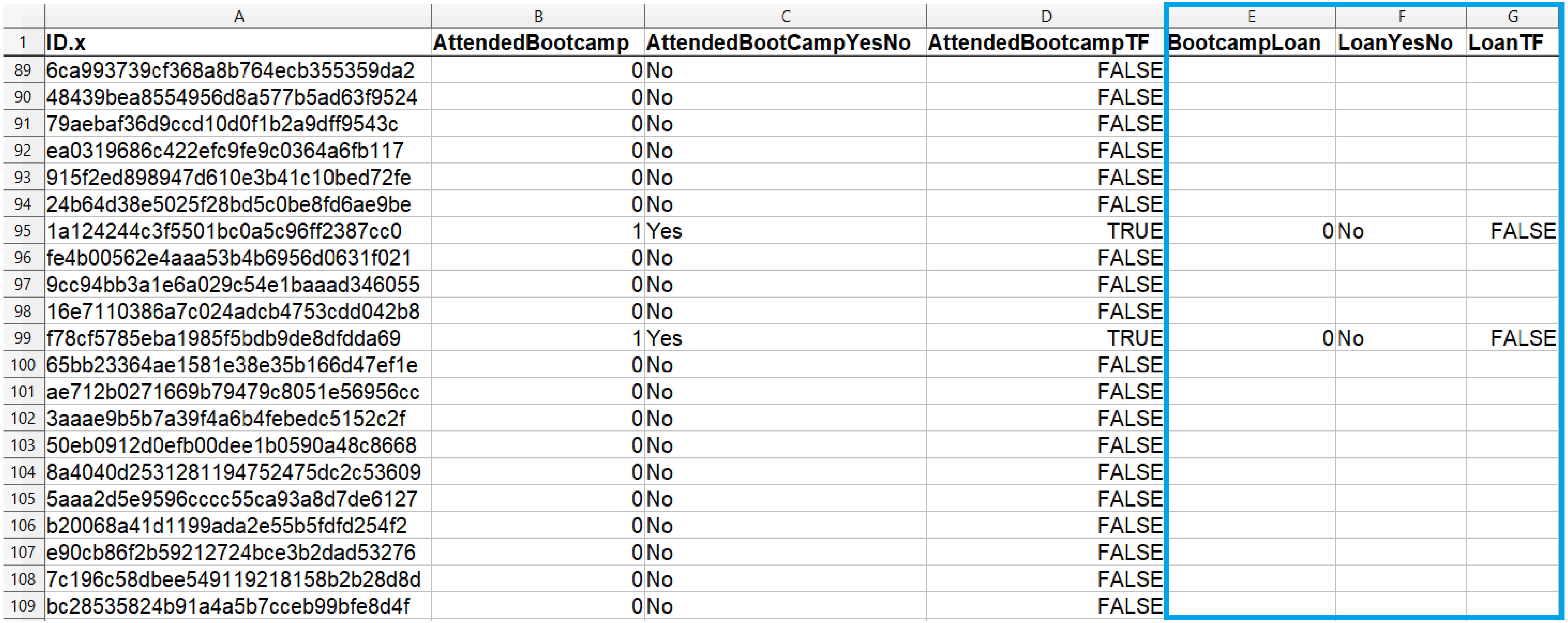
	A	B	C	D	E	F	G
1	**ID.x**	**AttendedBootcamp**	**AttendedBootCampYesNo**	**AttendedBootcampTF**	**BootcampLoan**	**LoanYesNo**	**LoanTF**
89	6ca993739cf368a8b764ecb355359da2	0	No	FALSE			
90	48439bea8554956d8a577b5ad63f9524	0	No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0	No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0	No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0	No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0	No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1	Yes	TRUE	0	No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0	No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0	No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0	No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1	Yes	TRUE	0	No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0	No	FALSE			
101	ae712b0271669b79479c8051e56956cc	0	No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0	No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0	No	FALSE			
104	8a4040d2531281194752475dc2c53609	0	No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0	No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0	No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0	No	FALSE			
108	7c196c58dbee549119218158b2b28d8d	0	No	FALSE			

Boolean Data



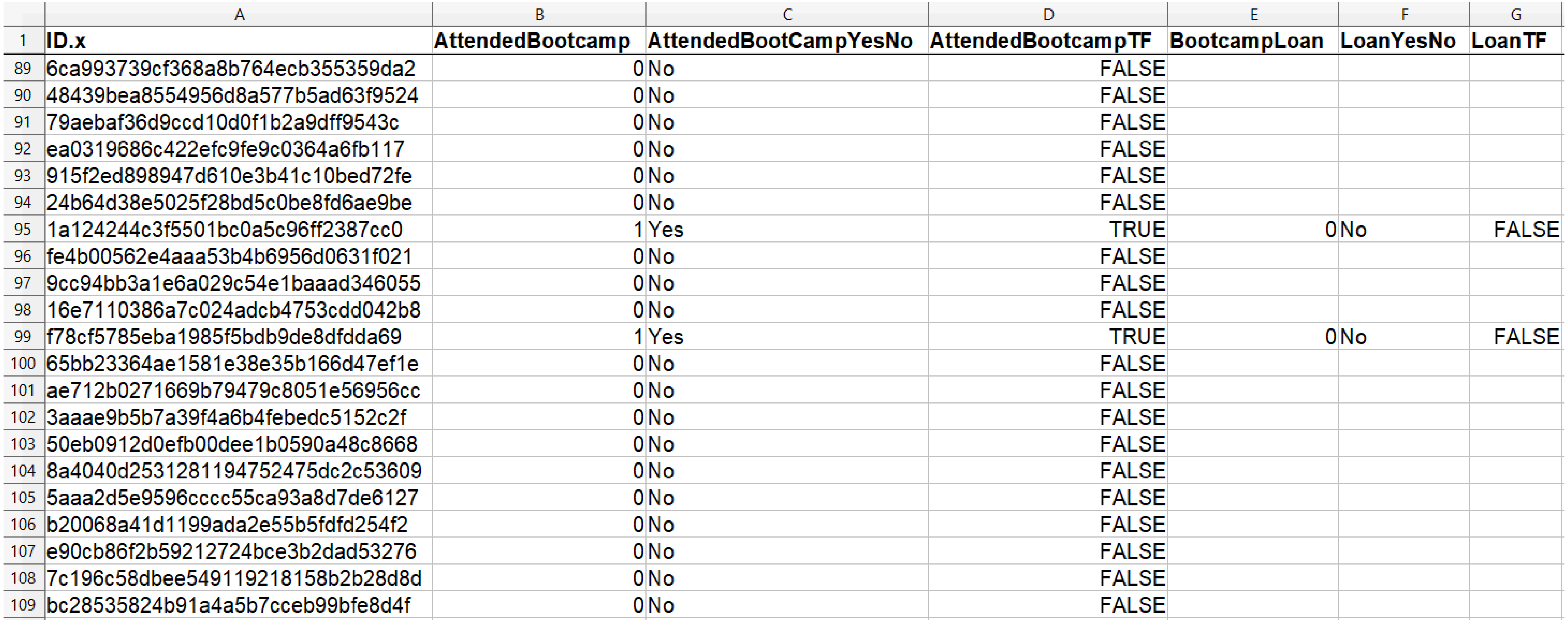
	A	B	C	D	E	F	G
1	**ID.x**	**AttendedBootcamp**	**AttendedBootCampYesNo**	**AttendedBootcampTF**	**BootcampLoan**	**LoanYesNo**	**LoanTF**
89	6ca993739cf368a8b764ecb355359da2	0	No	FALSE			
90	48439bea8554956d8a577b5ad63f9524	0	No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0	No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0	No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0	No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0	No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1	Yes	TRUE	0	No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0	No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0	No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0	No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1	Yes	TRUE	0	No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0	No	FALSE			
101	ae712b0271669b79479c8051e56956cc	0	No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0	No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0	No	FALSE			
104	8a4040d2531281194752475dc2c53609	0	No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0	No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0	No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0	No	FALSE			
108	7c196c58dbee549119218158b2b28d8d	0	No	FALSE			
109	bc28535824b91a4a5b7cceb99bfe8d4f	0	No	FALSE			

Boolean Data



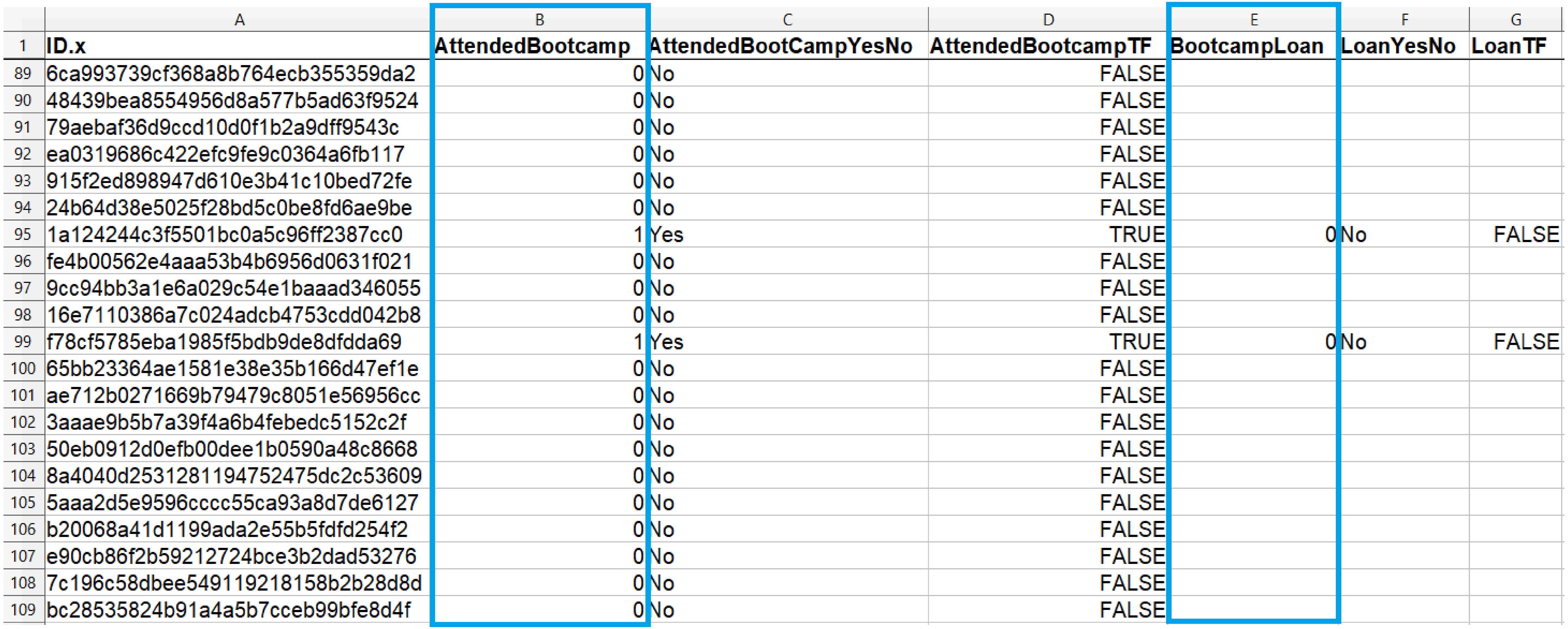
	A	B	C	D	E	F	G
1	**ID.x**	**AttendedBootcamp**	**AttendedBootCampYesNo**	**AttendedBootcampTF**	**BootcampLoan**	**LoanYesNo**	**LoanTF**
89	6ca993739cf368a8b764ecb355359da2	0	No	FALSE			
90	48439bea8554956d8a577b5ad63f9524	0	No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0	No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0	No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0	No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0	No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1	Yes	TRUE	0	No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0	No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0	No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0	No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1	Yes	TRUE	0	No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0	No	FALSE			
101	ae712b0271669b79479c8051e56956cc	0	No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0	No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0	No	FALSE			
104	8a4040d253128119475247 5dc2c53609	0	No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0	No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0	No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0	No	FALSE			
108	7c196c58dbee549119218158b2b28d8d	0	No	FALSE			
109	bc28535824b91a4a5b7cceb99bfe8d4f	0	No	FALSE			

Boolean Data



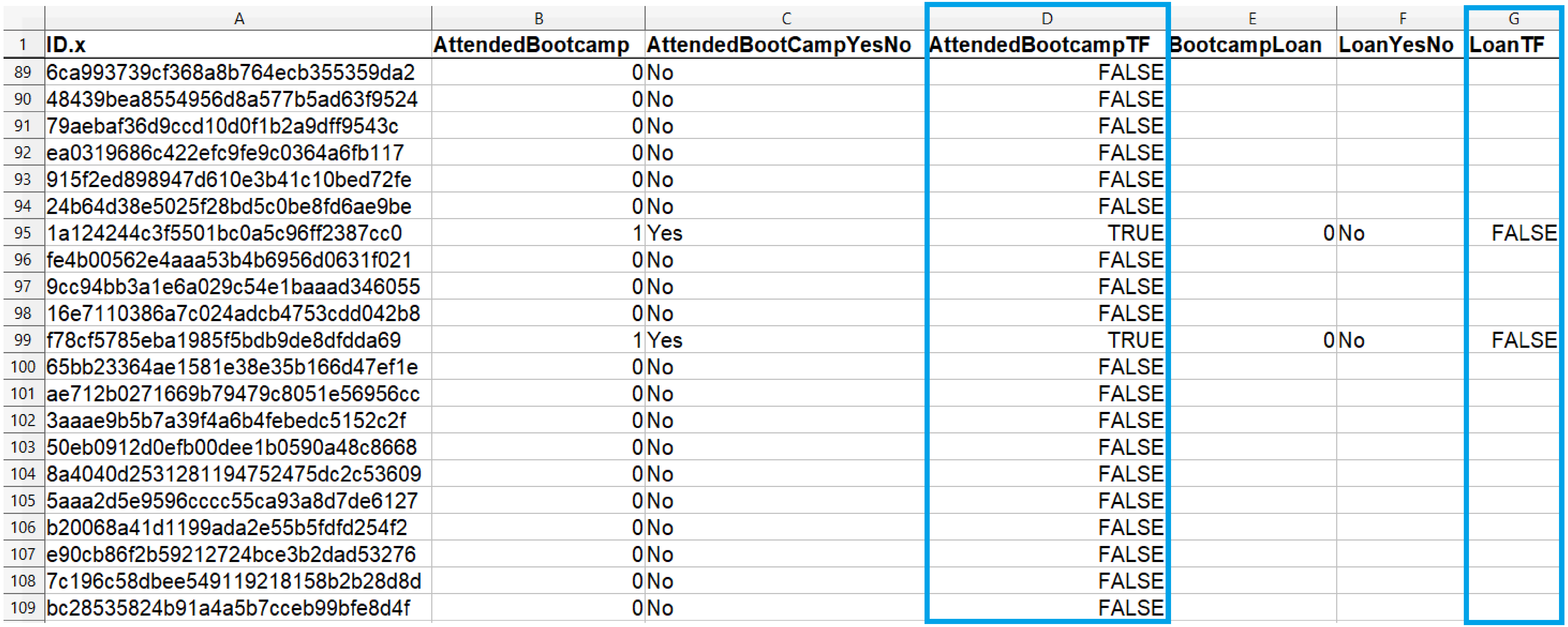
	A	B	C	D	E	F	G
1	**ID.x**	**AttendedBootcamp**	**AttendedBootCampYesNo**	**AttendedBootcampTF**	**BootcampLoan**	**LoanYesNo**	**LoanTF**
89	6ca993739cf368a8b764ecb355359da2	0	No	FALSE			
90	48439bea8554956d8a577b5ad63f9524	0	No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0	No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0	No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0	No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0	No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1	Yes	TRUE	0	No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0	No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0	No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0	No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1	Yes	TRUE	0	No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0	No	FALSE			
101	ae712b0271669b79479c8051e56956cc	0	No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0	No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0	No	FALSE			
104	8a4040d2531281194752475dc2c53609	0	No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0	No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0	No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0	No	FALSE			
108	7c196c58dbee549119218158b2b28d8d	0	No	FALSE			
109	bc28535824b91a4a5b7cceb99bfe8d4f	0	No	FALSE			

Boolean Data



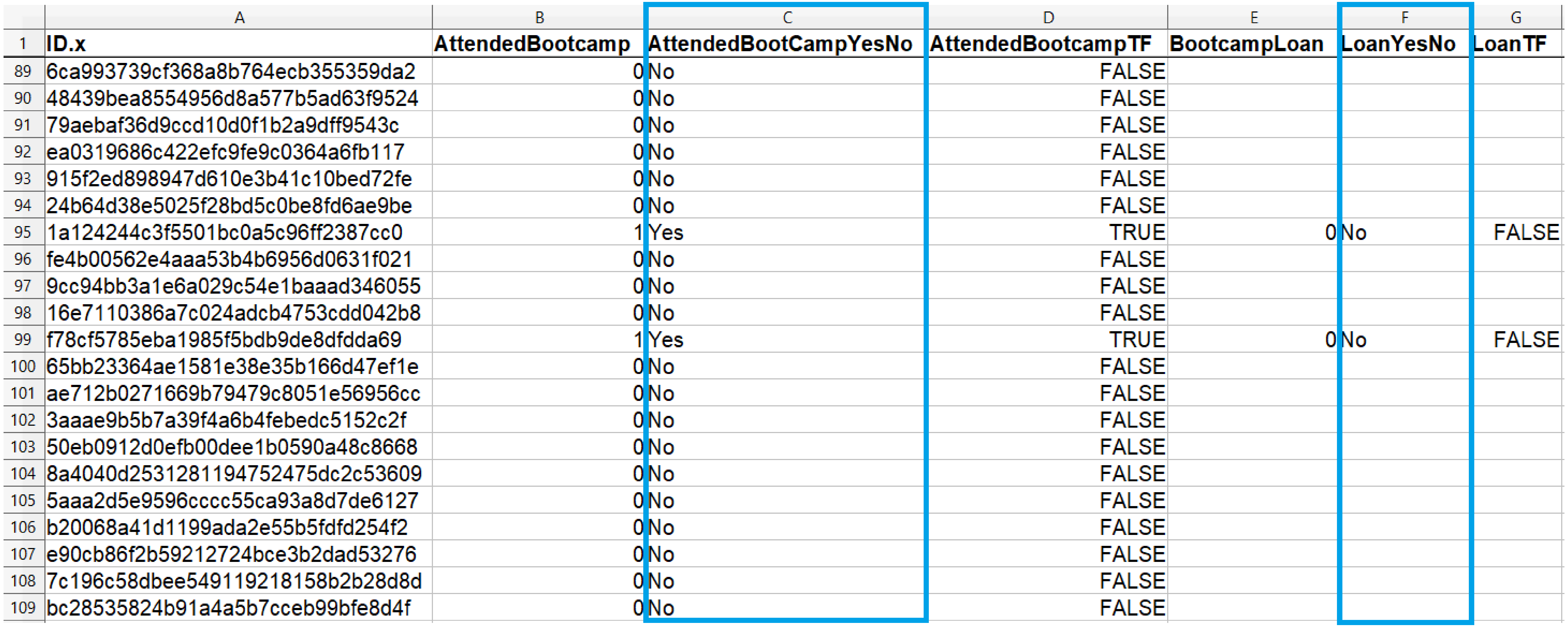
	A	B	C	D	E	F	G
1	**ID.x**	**AttendedBootcamp**	**AttendedBootCampYesNo**	**AttendedBootcampTF**	**BootcampLoan**	**LoanYesNo**	**LoanTF**
89	6ca993739cf368a8b764ecb355359da2	0	No	FALSE			
90	48439bea8554956d8a577b5ad63f9524	0	No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0	No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0	No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0	No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0	No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1	Yes	TRUE	0	No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0	No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0	No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0	No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1	Yes	TRUE	0	No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0	No	FALSE			
101	ae712b0271669b79479c8051e56956cc	0	No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0	No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0	No	FALSE			
104	8a4040d2531281194752475dc2c53609	0	No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0	No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0	No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0	No	FALSE			
108	7c196c58dbee549119218158b2b28d8d	0	No	FALSE			
109	bc28535824b91a4a5b7cceb99bfe8d4f	0	No	FALSE			

Boolean Data



	A	B	C	D	E	F	G
1	**ID.x**	**AttendedBootcamp**	**AttendedBootCampYesNo**	**AttendedBootcampTF**	**BootcampLoan**	**LoanYesNo**	**LoanTF**
89	6ca993739cf368a8b764ecb355359da2	0	No	FALSE			
90	48439bea8554956d8a577b5ad63f9524	0	No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0	No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0	No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0	No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0	No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1	Yes	TRUE	0	No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0	No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0	No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0	No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1	Yes	TRUE	0	No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0	No	FALSE			
101	ae712b0271669b79479c8051e56956cc	0	No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0	No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0	No	FALSE			
104	8a4040d2531281194752475dc2c53609	0	No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0	No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0	No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0	No	FALSE			
108	7c196c58dbee549119218158b2b28d8d	0	No	FALSE			
109	bc28535824b91a4a5b7cceb99bfe8d4f	0	No	FALSE			

Boolean Data



	A	B	C	D	E	F	G
1	ID.x	AttendedBootcamp	AttendedBootCampYesNo	AttendedBootcampTF	BootcampLoan	LoanYesNo	LoanTF
89	6ca993739cf368a8b764ecb355359da2	0	No	FALSE			
90	48439bea8554956d8a577b5ad63f9524	0	No	FALSE			
91	79aebaf36d9ccd10d0f1b2a9dff9543c	0	No	FALSE			
92	ea0319686c422efc9fe9c0364a6fb117	0	No	FALSE			
93	915f2ed898947d610e3b41c10bed72fe	0	No	FALSE			
94	24b64d38e5025f28bd5c0be8fd6ae9be	0	No	FALSE			
95	1a124244c3f5501bc0a5c96ff2387cc0	1	Yes	TRUE	0	No	FALSE
96	fe4b00562e4aaa53b4b6956d0631f021	0	No	FALSE			
97	9cc94bb3a1e6a029c54e1baaad346055	0	No	FALSE			
98	16e7110386a7c024adcb4753cdd042b8	0	No	FALSE			
99	f78cf5785eba1985f5bdb9de8dfdda69	1	Yes	TRUE	0	No	FALSE
100	65bb23364ae1581e38e35b166d47ef1e	0	No	FALSE			
101	ae712b0271669b79479c8051e56956cc	0	No	FALSE			
102	3aaae9b5b7a39f4a6b4febedc5152c2f	0	No	FALSE			
103	50eb0912d0efb00dee1b0590a48c8668	0	No	FALSE			
104	8a4040d2531281194752475dc2c53609	0	No	FALSE			
105	5aaa2d5e9596cccc55ca93a8d7de6127	0	No	FALSE			
106	b20068a41d1199ada2e55b5fdfd254f2	0	No	FALSE			
107	e90cb86f2b59212724bce3b2dad53276	0	No	FALSE			
108	7c196c58dbee549119218158b2b28d8d	0	No	FALSE			
109	bc28535824b91a4a5b7cceb99bfe8d4f	0	No	FALSE			

pandas and Booleans

```
bootcamp_data = pd.read_excel("fcc_survey_booleans.xlsx")
print(bootcamp_data.dtypes)
```

```
ID.x                         object
AttendedBootcamp            float64
AttendedBootCampYesNo        object
AttendedBootcampTF          float64
BootcampLoan                float64
LoanYesNo                    object
LoanTF                      float64
dtype: object
```

pandas and Booleans

```
# Count True values
print(bootcamp_data.sum())
```

```
AttendedBootcamp                38
AttendedBootcampTF              38
BootcampLoan                    14
LoanTF                          14
dtype: object
```

```
# Count NAs
print(bootcamp_data.isna().sum())
```

```
ID.x                          0
AttendedBootcamp              0
AttendedBootCampYesNo         0
AttendedBootcampTF            0
BootcampLoan                964
LoanYesNo                   964
LoanTF                      964
dtype: int64
```

```
# Load data, casting True/False columns as Boolean
bool_data = pd.read_excel("fcc_survey_booleans.xlsx",
                          dtype={"AttendedBootcamp": bool,
                                "AttendedBootCampYesNo": bool,
                                "AttendedBootcampTF":bool,
                                "BootcampLoan": bool,
                                "LoanYesNo": bool,
                                "LoanTF": bool})

print(bool_data.dtypes)
```

```
ID.x                      object
AttendedBootcamp            bool
AttendedBootCampYesNo       bool
AttendedBootcampTF          bool
BootcampLoan                bool
LoanYesNo                   bool
LoanTF                      bool
dtype: object
```

```
# Count True values
print(bool_data.sum())
```

```
AttendedBootcamp                             38
AttendedBootCampYesNo                      1000
AttendedBootcampTF                           38
BootcampLoan                                978
LoanYesNo                                  1000
LoanTF                                      978
dtype: object
```

```
# Count NA values
print(bool_data.isna().sum())
```

```
ID.x                        0
AttendedBootcamp            0
AttendedBootCampYesNo       0
AttendedBootcampTF          0
BootcampLoan                0
LoanYesNo                   0
LoanTF                      0
dtype: int64
```

pandas and Booleans

- `pandas` loads `True` / `False` columns as float data by default
- Specify a column should be `bool` with `read_excel()` 's `dtype` argument
- Boolean columns can *only* have `True` and `False` values
- **NA/missing values in Boolean columns are changed to `True`**
- `pandas` automatically recognizes some values as `True` / `False` in Boolean columns
- Unrecognized values in a Boolean column are also changed to `True`

Setting Custom True/False Values

- Use `read_excel()` 's `true_values` argument to set custom `True` values
- Use `false_values` to set custom `False` values
- Each takes a list of values to treat as `True` / `False` , respectively
- Custom `True` / `False` values are only applied to columns set as Boolean

Setting Custom True/False Values

```
# Load data with Boolean dtypes and custom T/F values
bool_data = pd.read_excel("fcc_survey_booleans.xlsx",
                          dtype={"AttendedBootcamp": bool,
                                "AttendedBootCampYesNo": bool,
                                "AttendedBootcampTF":bool,
                                "BootcampLoan": bool,
                                "LoanYesNo": bool,
                                "LoanTF": bool},
                          true_values=["Yes"],
                          false_values=["No"])
```

Setting Custom True/False Values

```
print(bool_data.sum())
```

```
AttendedBootcamp                                  38
AttendedBootCampYesNo                             38
AttendedBootcampTF                                38
BootcampLoan                                     978
LoanYesNo                                        978
LoanTF                                           978
dtype: object
```

Boolean Considerations

- Are there missing values, or could there be in the future?
- How will this column be used in analysis?
- What would happen if a value were incorrectly coded as `True` ?
- Could the data be modeled another way (e.g., as floats or integers)?

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS

Modifying imports: parsing dates

STREAMLINED DATA INGESTION WITH PANDAS

Amany Mahfouz
Instructor

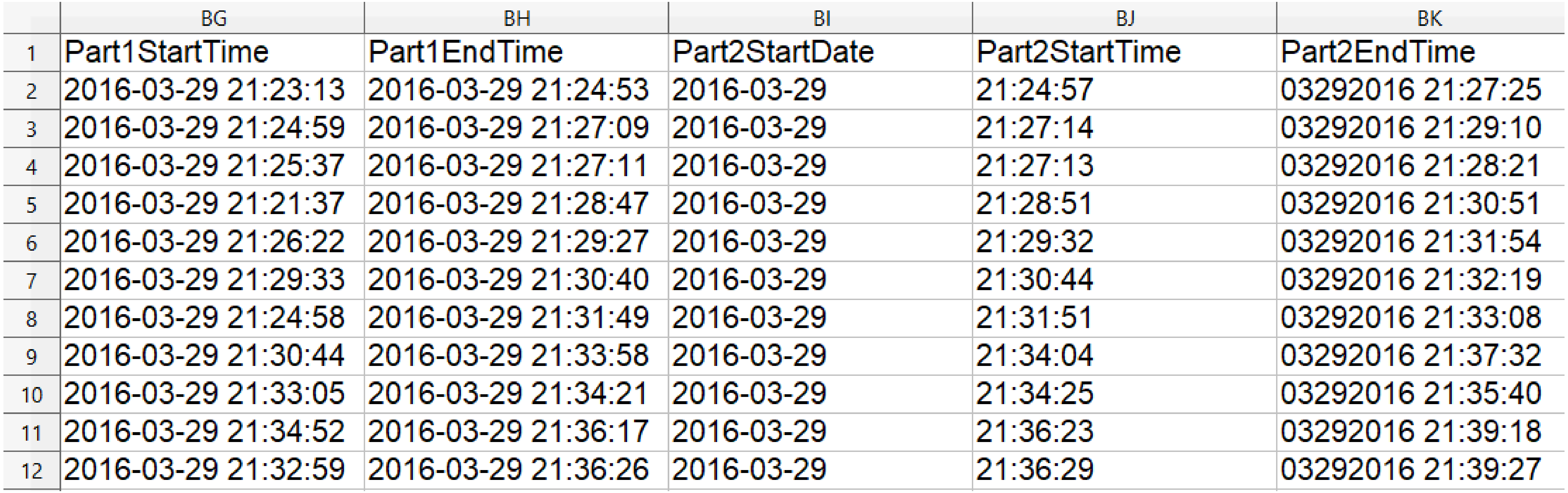
Date and Time Data

- Dates and times have their own data type and internal representation
- Datetime values can be translated into string representations
- Common set of codes to describe datetime string formatting

pandas and Datetimes

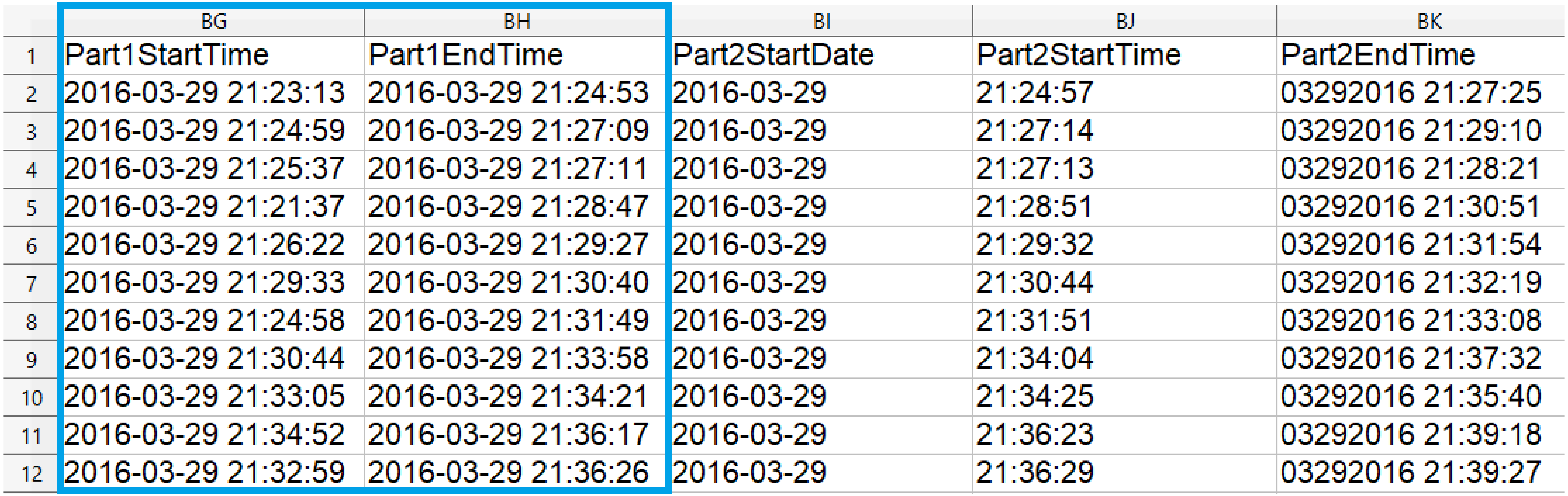
- Datetime columns are loaded as objects (strings) by default
- Specify that columns have datetimes with the `parse_dates` argument (not `dtype` !)
- `parse_dates` can accept:
    - a list of column names or numbers to parse
    - a list containing lists of columns to combine and parse
    - a dictionary where keys are new column names and values are lists of columns to parse together

pandas and Datetimes



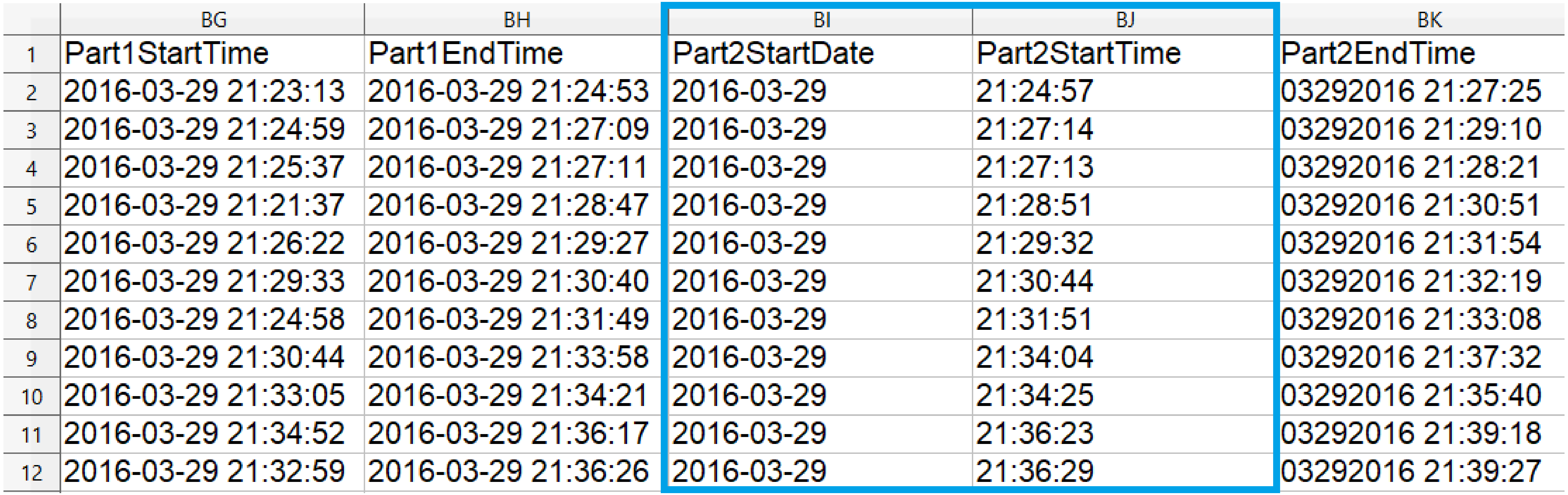
	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

pandas and Datetimes



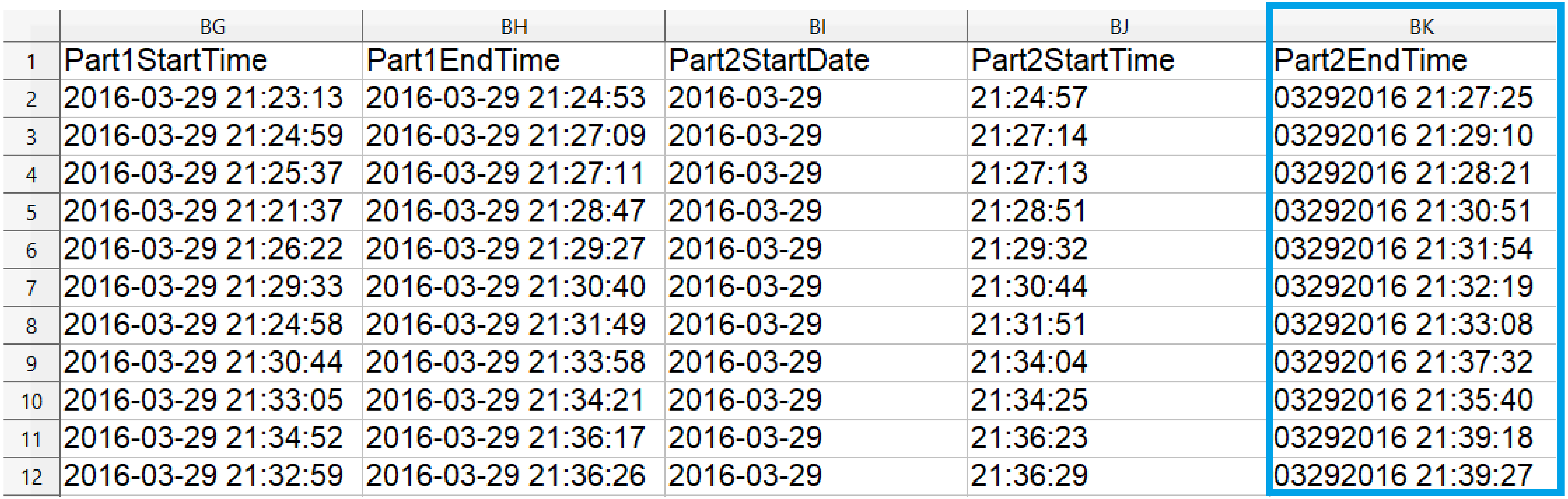
	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

pandas and Datetimes



	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

pandas and Datetimes



	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

Parsing Dates

```
# List columns of dates to parse
date_cols = ["Part1StartTime", "Part1EndTime"]


# Load file, parsing standard datetime columns
survey_df = pd.read_excel("fcc_survey.xlsx",
                          parse_dates=date_cols)
```

Parsing Dates

```
# Check data types of timestamp columns
print(survey_df[["Part1StartTime",
                 "Part1EndTime",
                 "Part2StartDate",
                 "Part2StartTime",
                 "Part2EndTime"]].dtypes)
```

```
Part1StartTime    datetime64[ns]
Part1EndTime      datetime64[ns]
Part2StartDate            object
Part2StartTime            object
Part2EndTime              object
dtype: object
```

Parsing Dates

```
# List columns of dates to parse
date_cols = ["Part1StartTime",
             "Part1EndTime",
             [["Part2StartDate", "Part2StartTime"]]]
# Load file, parsing standard and split datetime columns
survey_df = pd.read_excel("fcc_survey.xlsx",
                          parse_dates=date_cols)
print(survey_df.head(3))
```

```
  Part2StartDate_Part2StartTime   Age        ...    SchoolMajor  StudentDebtOwe
0           2016-03-29 21:24:57  28.0        ...            NaN           20000
1           2016-03-29 21:27:14  22.0        ...            NaN             NaN
2           2016-03-29 21:27:13  19.0        ...            NaN             NaN

[3 rows x 98 columns]
```

Parsing Dates

```
# List columns of dates to parse
date_cols = {"Part1Start": "Part1StartTime",
             "Part1End": "Part1EndTime",
             "Part2Start": ["Part2StartDate",
                            "Part2StartTime"]}

# Load file, parsing standard and split datetime columns
survey_df = pd.read_excel("fcc_survey.xlsx",
                          parse_dates=date_cols)

print(survey_df.Part2Start.head(3))
```

```
0   2016-03-29 21:24:57
1   2016-03-29 21:27:14
2   2016-03-29 21:27:13
Name: Part2Start, dtype: datetime64[ns]
```

Non-Standard Dates

- `parse_dates` doesn't work with non-standard datetime formats
- Use `pd.to_datetime()` after loading data if `parse_dates` won't work
- `to_datetime()` arguments:
    - Data frame and column to convert
    - `format` : string representation of datetime format

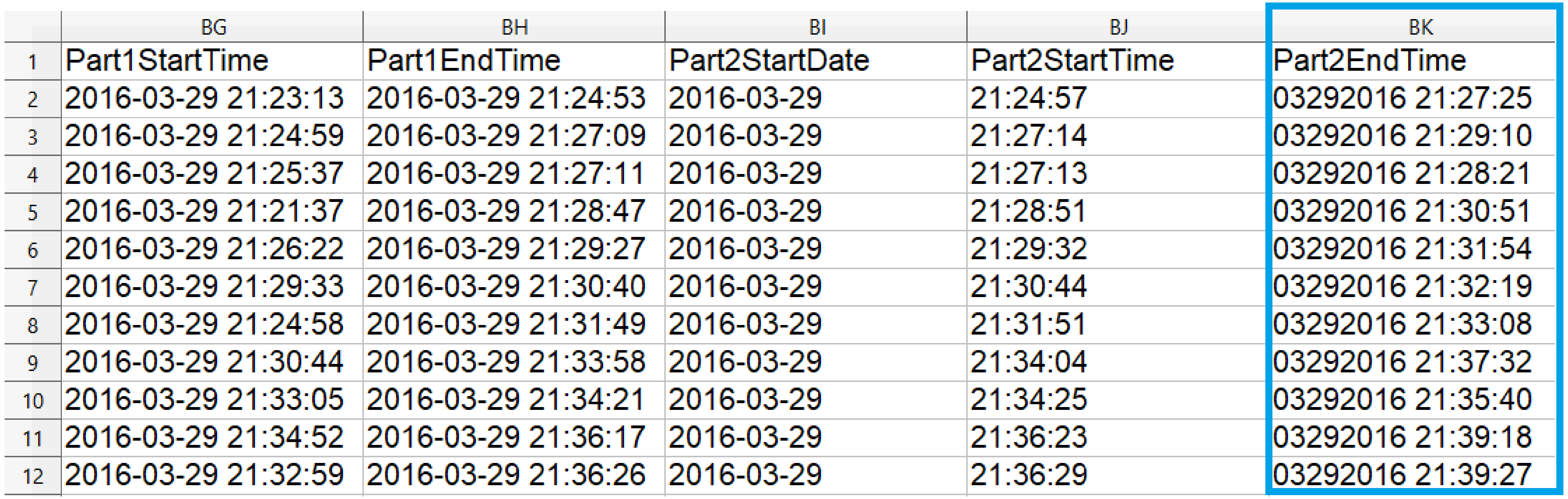
Datetime Formatting

- Describe datetime string formatting with codes and characters
- Refer to **strftime.org** for the full list

Datetime Formatting

Code	Meaning	Example
`%Y`	Year (4-digit)	1999
`%m`	Month (zero-padded)	03
`%d`	Day (zero-padded)	01
`%H`	Hour (24-hour clock)	21
`%M`	Minute (zero-padded)	09
`%S`	Second (zero-padded)	05

Parsing Non-Standard Dates



	BG	BH	BI	BJ	BK
1	Part1StartTime	Part1EndTime	Part2StartDate	Part2StartTime	Part2EndTime
2	2016-03-29 21:23:13	2016-03-29 21:24:53	2016-03-29	21:24:57	03292016 21:27:25
3	2016-03-29 21:24:59	2016-03-29 21:27:09	2016-03-29	21:27:14	03292016 21:29:10
4	2016-03-29 21:25:37	2016-03-29 21:27:11	2016-03-29	21:27:13	03292016 21:28:21
5	2016-03-29 21:21:37	2016-03-29 21:28:47	2016-03-29	21:28:51	03292016 21:30:51
6	2016-03-29 21:26:22	2016-03-29 21:29:27	2016-03-29	21:29:32	03292016 21:31:54
7	2016-03-29 21:29:33	2016-03-29 21:30:40	2016-03-29	21:30:44	03292016 21:32:19
8	2016-03-29 21:24:58	2016-03-29 21:31:49	2016-03-29	21:31:51	03292016 21:33:08
9	2016-03-29 21:30:44	2016-03-29 21:33:58	2016-03-29	21:34:04	03292016 21:37:32
10	2016-03-29 21:33:05	2016-03-29 21:34:21	2016-03-29	21:34:25	03292016 21:35:40
11	2016-03-29 21:34:52	2016-03-29 21:36:17	2016-03-29	21:36:23	03292016 21:39:18
12	2016-03-29 21:32:59	2016-03-29 21:36:26	2016-03-29	21:36:29	03292016 21:39:27

```
format_string = "%m%d%Y %H:%M:%S"
survey_df["Part2EndTime"] = pd.to_datetime(survey_df["Part2EndTime"],
                                           format=format_string)
```

Parsing Non-Standard Dates

```
print(survey_df.Part2EndTime.head())
```

```
0   2016-03-29 21:27:25
1   2016-03-29 21:29:10
2   2016-03-29 21:28:21
3   2016-03-29 21:30:51
4   2016-03-29 21:31:54
Name: Part2EndTime, dtype: datetime64[ns]
```

Let's practice!

STREAMLINED DATA INGESTION WITH PANDAS