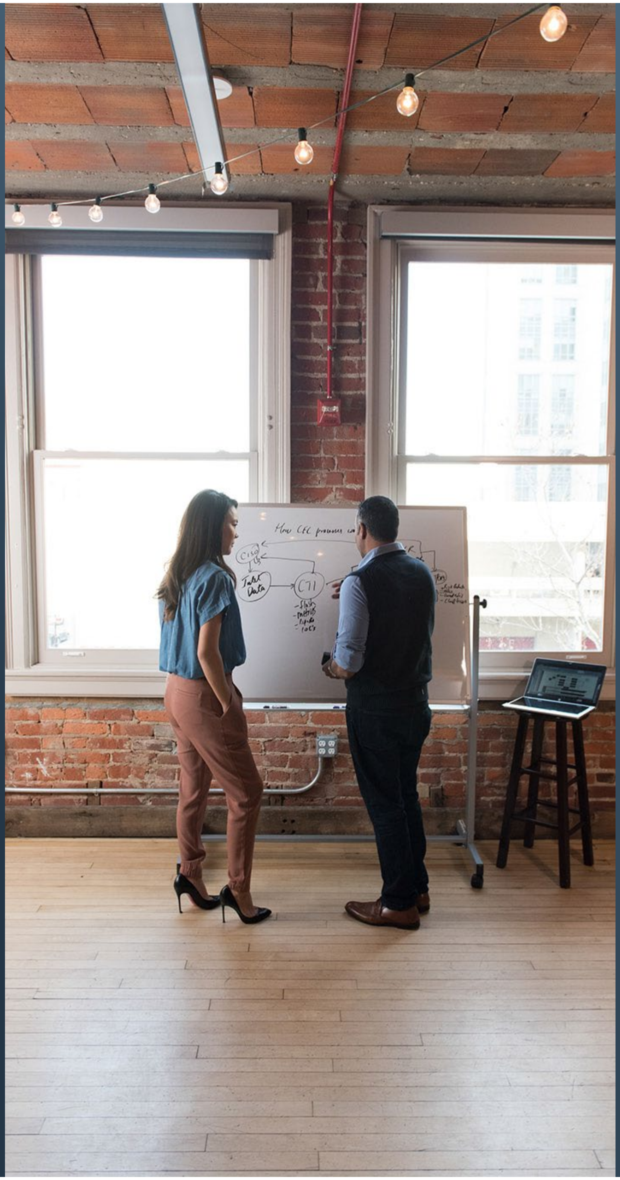
ICEAA International Cost Estimating and Analysis Association

Booz | Allen | Hamilton



DATA CLEANING IN PYTHON FOR BEGINNERS

Alexis Somers, CCE/A

ICEAA Workshop 2024

MAY 2024

CONSULTING | ANALYTICS | DIGITAL SOLUTIONS | ENGINEERING | CYBER

AGENDA

INTRODUCTION & FIRST STEPS

DATA TYPES

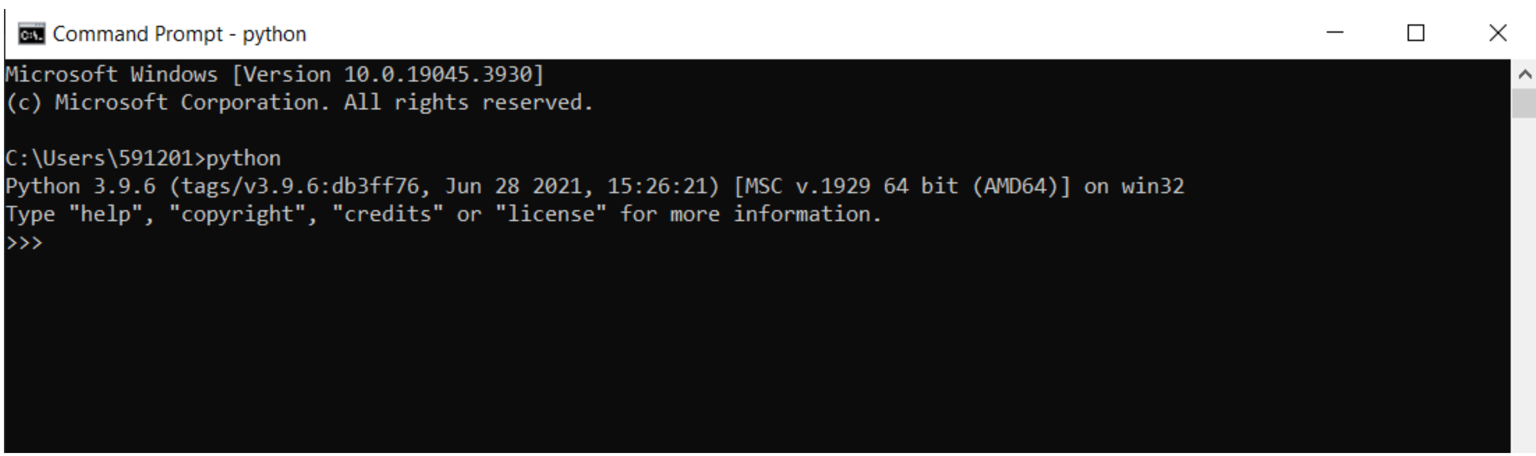
USING PANDAS

EXAMPLE - PRICING TABLES

PYTHON INTRODUCTION

- Python is a popular, powerful programming language that is easy to learn and easy to use
- Commonly used for developing websites and software, task automation, data analysis, and data visualization
- Open source, so anyone can contribute to its development
- Code that is as understandable as plain English
- Suitable for everyday tasks, allowing for short development times

Standard Python Prompt (Windows)

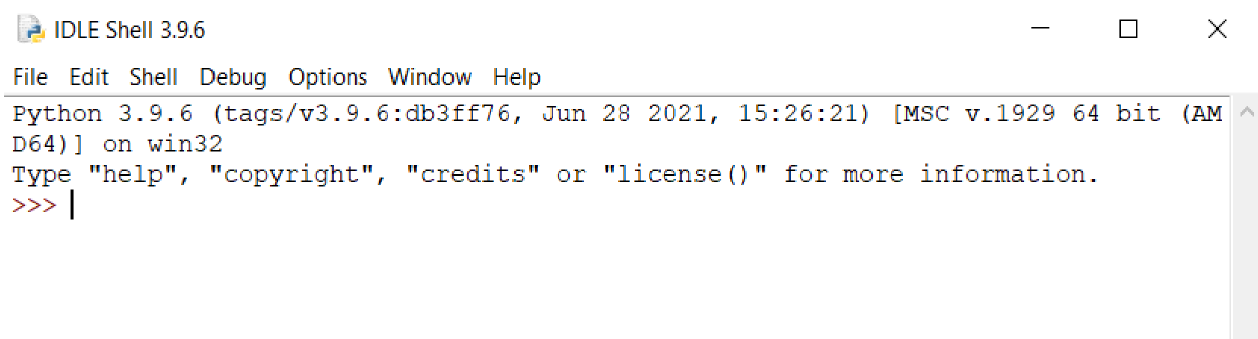


ADVANTAGES OVER EXCEL

- Reproducibility
  - Saves a tremendous amount of time and guarantees consistency
  - Making changes to a dataset that is then updated elsewhere and re-provided to you
  - In a script all the changes are documented and you can add comments explaining the steps and the reasoning
- Faster and more powerful
- Easier than VBA
- Ability to automate data prep in specific data environments
  - e.g. Advana, Tableau

FIRST STEPS

- Install Python (visit python.org)
- Choose an Integrated Development Environment (IDE) or text editor
  - Many are tailored specifically to make Python editing easy
- Popular options: IDLE, Spyder, PyCharm, Jupyter
- Anaconda is a distribution of Python that includes packages, IDEs and package manager tools for programming in Python



PYTHON LIBRARIES

- Python libraries are collections of reusable code modules that you can integrate into your projects to save time and effort

Pandas Data analysis and manipulation	**NumPy** Mathematical functions
Matplotlib Data visualisations	**SeaBorn** Data visualisations
Tensorflow Machine Learning	**Keras** Deep Learning
SciPy Scientific computing	**PyTorch** Machine Learning
Scrapy Web crawling	**SQLModel** Interact with SQL databases

THINGS TO NOTE/REMEMBER

- Python begins counting at zero
- Python uses new lines to complete a command
- # is the Python comment character

```
# This is a comment
import os
import pandas as pd # You can rename the module, if you want
```

```
2+3 # This will not print, be aware when it will and when it won't.
a = 2 + 3
print(a) # The print statement prints whatever you put in it
print(2+3)
```

```
5
5
```

AGENDA

DATA TYPES

Data Types	Classes	Description
Numeric	int, float, complex	holds numeric values
String	str	holds sequence of characters
Sequence	list, tuple, range	holds collection of items
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either True or False
Set	set, frozenset	hold collection of unique items

KEY DATA TYPES – INTEGER, FLOAT, STRING

```
a = 6 # an integer
b = 6.5 # a float
c = '67' # a string
d = 'ICEAA'
### e = Alexis ## This won't work
print(a + b)
print(c + d) # will concatenate the strings
print(a + c) # you can't combine different types
```

```
12.5
67ICEAA
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[25], line 8
      6 print(a + b)
      7 print(c + d) # will concatenate the strings
----> 8 print(a + c) # you can't combine different types

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

KEY DATA TYPES – LIST

LISTS

- List items are ordered, changeable, and allow duplicate values
- List items are indexed. First item has index [0], second item has index [1] etc.

```
my_list = ['Chicken', 'Beef', 'Pork'] # A list holds objects
print(my_list)
my_list.append('Tofurkey')
print(my_list) # The list will now contain 'Tofurkey' at the end
# A list can do a lot
# List comprehensions are amazing
# But this isn't for here
```

```
['Chicken', 'Beef', 'Pork']
['Chicken', 'Beef', 'Pork', 'Tofurkey']
```

AGENDA

INTRODUCTION & FIRST STEPS

DATA TYPES

USING PANDAS

EXAMPLE - PRICING TABLES

USING PANDAS: FIRST STEPS

- Import dataset
  - Navigate to the directory where your dataset is stored
  - Determine the file type in order to determine the pandas command
    - For Excel files, pd.read_excel(file_name) (Note: requires an additional package to run)
    - For .csv files, pd.read_csv(file_name)
- Notes:
  - For Excel files, make sure to specify the sheet name pd.read_excel(file_name, sheet_name=0)

```
os.chdir('c:\\Users\\username\\documents\\')
```

```
# Pandas adds two primary types
# Data Frames (matrices)
# Series (columns)
# They are basically augmented lists
iceaa = pd.read_excel('Sample_Dataset.xlsx', sheet_name = 'Spend Plan')
```

USING PANDAS (CONT.)

```
# Can view the sheet in full
print(iceaa)
```

```
    ID         Name       F_U  ...   Travel      ODC  TotWYCost
0    1    Install A    Funded  ...      NaN      NaN  184800.00
1    2    Install B    Funded  ...      NaN      NaN   20000.00
2    3   Software A    Funded  ...  15000.0      0.0  360000.00
3    4   Software B  Unfunded  ...      0.0      0.0  170000.00
4    5   Software C  Unfunded  ...      0.0      0.0   45000.00
..  ..          ...       ...  ...      ...      ...        ...
10  11  Personnel A    Funded  ...      0.0      0.0  137649.61
11  12  Personnel B    Funded  ...      0.0      0.0   70000.00
12  13  Personnel C    Funded  ...      0.0      0.0  209204.30
13  14  Personnel D    Funded  ...      0.0  53616.0   53616.00
14  15  Personnel E    Funded  ...      NaN      NaN       0.00

[15 rows x 15 columns]
```

USING PANDAS (CONT.)

```
# Can view the beginning of the file
print(iceaa.head())
# .head() can customize the number of beginning rows to show
```

```
   ID        Name       F_U  ...   Travel  ODC TotWYCost
0   1   Install A    Funded  ...      NaN  NaN  184800.0
1   2   Install B    Funded  ...      NaN  NaN   20000.0
2   3  Software A    Funded  ...  15000.0  0.0  360000.0
3   4  Software B  Unfunded  ...      0.0  0.0  170000.0
4   5  Software C  Unfunded  ...      0.0  0.0   45000.0

[5 rows x 15 columns]
```

```
# Can view the "shape": (# of rows, # of columns)
print(iceaa.shape)
# Can view the columns only
print(iceaa.columns)
```

```
(15, 15)
Index(['ID', 'Name', 'F_U', 'APPN', 'LineItem', 'GWBSDef', 'ContractNum', 'POPStart', 'P
OPEnd',
       'GovtWY', 'KTRWY', 'LaborCost', 'Travel', 'ODC', 'TotWYCost'],
      dtype='object')
```

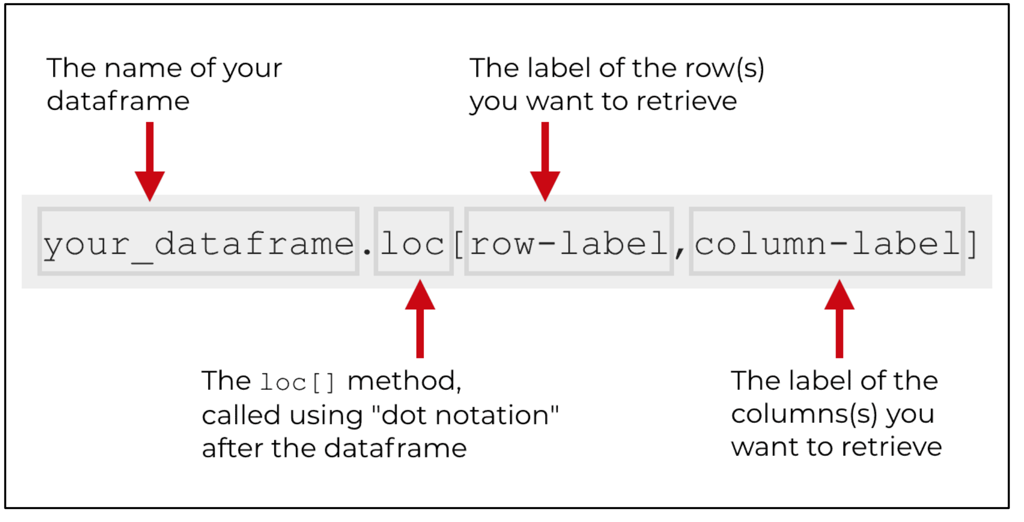
USING PANDAS (CONT.)

```
# Can view the types that each column is
print(iceaa.dtypes)
```

```
ID              int64
Name           object
F_U            object
APPN           object
LineItem       object
               ...
KTRWY         float64
LaborCost     float64
Travel        float64
ODC           float64
TotWYCost     float64
Length: 15, dtype: object
```

USING PANDAS: LOCATORS

- .loc[] is an important method used for accessing a group of rows and columns
- Written as .loc[rows, columns]
  - rows is usually a logical statement
  - columns is either a string (one column) or a list of columns
- .loc[] is label-based, meaning you specify rows and columns based on their row and column labels
  - Also an option to use .iloc[], which is integer position-based, meaning you specify rows and columns by their integer position values (0-based integer position)



USING PANDAS: LOCATORS (CONT.)

```
# Editing will mostly be done using the .loc[] method
# and Boolean logic
# This will fetch all rows (that's the :)
#
print(iceaa.loc[:,'Name'])
print(iceaa.loc[:, ['Name', 'ContractNum']])
# The "\n" is a control code for 'new line' -- check the data!
```

```
0       Install A
1       Install B
2      Software A
3      Software B
4      Software C
          ...
10    Personnel A
11    Personnel B
12    Personnel C
13    Personnel D
14    Personnel E
Name: Name, Length: 15, dtype: object
```

```
           Name                                ContractNum
0     Install A  Contract: N0012345D1234; \nDO: N0001234G1234
1     Install B  Contract: N0012345D1234; \nDO: N0001234G1234
2    Software A                          N12345-12-D-1234
3    Software B                              N123459D1234
4    Software C                              N123459D1235
..          ...                                        ...
10  Personnel A                              N123456D1234
11  Personnel B                              N123456D1234
12  Personnel C                                       NaN
13  Personnel D                                       NaN
14  Personnel E                                       NaN

[15 rows x 2 columns]
```

USING PANDAS: LOCATORS (CONT.)

```
# Specifying rows requires logic
# The following will give me only rows where the value for "F_U" is 'Funded'
print(iceaa.loc[iceaa['F_U'] == 'Funded', :])
```

```
    ID          Name     F_U  ...    Travel      ODC  TotWYCost
0    1     Install A  Funded  ...       NaN      NaN  184800.00
1    2     Install B  Funded  ...       NaN      NaN   20000.00
2    3    Software A  Funded  ...  15000.00      0.0  360000.00
5    6     License A  Funded  ...  12594.87      0.0  137719.87
6    7     License B  Funded  ...       NaN      0.0   82589.77
..  ..           ...     ...  ...       ...      ...        ...
10  11   Personnel A  Funded  ...      0.00      0.0  137649.61
11  12   Personnel B  Funded  ...      0.00      0.0   70000.00
12  13   Personnel C  Funded  ...      0.00      0.0  209204.30
13  14   Personnel D  Funded  ...      0.00  53616.0   53616.00
14  15   Personnel E  Funded  ...       NaN      NaN       0.00

[13 rows x 15 columns]
```

USING PANDAS: LOCATORS (CONT.)

```
print(iceaa.loc[:, 'F_U'].value_counts())
```

```
F_U
Funded      13
Unfunded     2
Name: count, dtype: int64
```

```
print(iceaa.loc[:, 'Name'].unique())
```

```
['Install A' 'Install B' 'Software A' 'Software B' 'Software C'
 'License A' 'License B' 'License C' 'Software D' 'Software E'
 'Personnel A' 'Personnel B' 'Personnel C' 'Personnel D' 'Personnel E']
```

```
# The following will give me the Name for rows where the values for "APPN" is either 'OPN' or "RDTE"
print(iceaa.loc[iceaa['APPN'].isin(['OPN', 'RDTE']), 'Name'])
```

```
0        Install A
1        Install B
2       Software A
3       Software B
4       Software C
5        License A
6        License B
7        License C
10     Personnel A
11     Personnel B
Name: Name, dtype: object
```

USING PANDAS: STRINGS

- Columns will often contain many different string entries
- It is useful to access specific string information for some/all entries

```
# This gives the first two letters of each entry
print(iceaa.loc[:, 'Name'].str[:2])
```

```
0     In
1     In
2     So
3     So
4     So
      ..
10    Pe
11    Pe
12    Pe
13    Pe
14    Pe
Name: Name, Length: 15, dtype: object
```

USING PANDAS: STRINGS (CONT.)

```
# This returns a true/false for entries which contain the word "software"
print(iceaa.loc[:, 'Name'].str.contains('software'))
# A bit useless on its own.
```

```
0     False
1     False
2     False
3     False
4     False
      ...
10    False
11    False
12    False
13    False
14    False
Name: Name, Length: 15, dtype: bool
```

```
# But put it with a location accessor
print(iceaa.loc[iceaa['Name'].str.contains('software|Software'), ['Name', 'POPStart', 'POPEnd']])
# Within the string, the | is an "or"
# It'll find entries with "software" or "Software"
```

```
         Name   POPStart     POPEnd
2  Software A 2023-10-23 2025-01-25
3  Software B        NaT        NaT
4  Software C 2023-10-23 2025-01-25
8  Software D 2024-02-24 2024-10-24
9  Software E 2024-02-24 2024-10-24
```

USING PANDAS: ADDING INFORMATION

```
# Just declare a new column name and give it a value
# Lets make a new column that tells us if the cost is "large", which we'll define as over 150000
iceaa['large_cost'] = 'N'
# I find it helps to give a value that is either the baseline or can't happen, to make sure you did it right
print(iceaa)
```

```
    ID         Name       F_U  ...      ODC   TotWYCost large_cost
0    1    Install A    Funded  ...      NaN   184800.00          N
1    2    Install B    Funded  ...      NaN    20000.00          N
2    3   Software A    Funded  ...      0.0   360000.00          N
3    4   Software B  Unfunded  ...      0.0   170000.00          N
4    5   Software C  Unfunded  ...      0.0    45000.00          N
..  ..          ...       ...  ...      ...         ...        ...
10  11  Personnel A    Funded  ...      0.0   137649.61          N
11  12  Personnel B    Funded  ...      0.0    70000.00          N
12  13  Personnel C    Funded  ...      0.0   209204.30          N
13  14  Personnel D    Funded  ...  53616.0    53616.00          N
14  15  Personnel E    Funded  ...      NaN        0.00          N

[15 rows x 16 columns]
```

USING PANDAS: ADDING INFORMATION (CONT.)

```
iceaa.loc[iceaa['TotWYCost'] >= 150000, 'large_cost'] = 'Y'
print(iceaa)
```

```
    ID         Name       F_U  ...      ODC  TotWYCost large_cost
0    1    Install A    Funded  ...      NaN  184800.00          Y
1    2    Install B    Funded  ...      NaN   20000.00          N
2    3   Software A    Funded  ...      0.0  360000.00          Y
3    4   Software B  Unfunded  ...      0.0  170000.00          Y
4    5   Software C  Unfunded  ...      0.0   45000.00          N
..  ..          ...       ...  ...      ...        ...        ...
10  11  Personnel A    Funded  ...      0.0  137649.61          N
11  12  Personnel B    Funded  ...      0.0   70000.00          N
12  13  Personnel C    Funded  ...      0.0  209204.30          Y
13  14  Personnel D    Funded  ...  53616.0   53616.00          N
14  15  Personnel E    Funded  ...      NaN       0.00          N

[15 rows x 16 columns]
```

```
print(iceaa.loc[:, ['POPStart', 'POPEnd']])
```

```
     POPStart     POPEnd
0  2022-10-22 2022-12-23
1  2024-02-24 2025-01-25
2  2023-10-23 2025-01-25
3         NaT        NaT
4  2023-10-23 2025-01-25
..        ...        ...
10        NaT        NaT
11        NaT        NaT
12 2024-02-24 2025-01-25
13 2023-10-23 2024-10-24
14 2023-10-23 2024-10-24

[15 rows x 2 columns]
```

OTHER BASIC FUNCTIONS/METHODS IN PANDAS

- .unique()
  - Lists all unique entries in a column
- .value_counts()
  - Lists the unique values and the number of times they appear in a column
- .to_numeric()
  - Converts a column to a numeric type (integer or float)
- .to_datetime()
  - Converts a column to datetime format (e.g. 2024-05-12)
- .isna()
  - Says whether or not an entry is a missing value
- .fillna()
  - Fills all missing values with specified value (e.g. df[col].fillna(1) fills missing values with 1)
- .dt accessor
  - When manipulating a datetime type, df[col].dt.year gives the year; df[col].dt.month gives the month, etc.

USING PANDAS (CONT.)

```
print(iceaa.loc[:, 'POPStart'].dt.month)
print(iceaa.loc[:, 'POPStart'].dt.year)
```

```
0     10.0
1      2.0
2     10.0
3      NaN
4     10.0
      ...
10     NaN
11     NaN
12     2.0
13    10.0
14    10.0
Name: POPStart, Length: 15, dtype: float64
0     2022.0
1     2024.0
2     2023.0
3        NaN
4     2023.0
       ...
10       NaN
11       NaN
12    2024.0
13    2023.0
14    2023.0
Name: POPStart, Length: 15, dtype: float64
```

AGENDA

INTRODUCTION & FIRST STEPS

DATA TYPES

USING PANDAS

EXAMPLE - PRICING TABLES

EXAMPLE – PRICING TABLES

- Client/Vendor provides you the following information:

Qty Procured	2024	2025	2026	2027	2028
	First Award Only (At Award)	13-24 Months After	25-36 Months After	37-48 Months After	49-60 Months After
1	$ 649,699	$ 275,373	$ 285,074	$ 268,306	$ 273,934

- Notes:
  - For every 10 additional radios purchased, a bulk buy discount of 19% applies.
  - After 50 radios, the bulk buy discount increases by 0.6% for each additional radio.

- Desired changes to data:
  - Fill in additional rows and columns for pricing table
  - Adjust for bulk buy discounts
  - Calculate escalation factor
  - Add columns for 4 additional years of estimates

EXAMPLE - SAMPLE DATA

Radios									
Escalation	1.022004507								
Qty Procured	2024	2025	2026	2027	2028	2029	2030	2031	2032
	First Award Only (At Award)	13-24 Months After	25-36 Months After	37-48 Months After	49-60 Months After	Extrap	Extrap	Extrap	Extrap
1	$ 649,699	$ 275,373	$ 285,074	$ 268,036	$ 273,934				
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									
19									

EXAMPLE (CONT.)

```
iceaa_p = pd.read_excel('Sample_Dataset.xlsx', sheet_name = 'Pricing Tables')
```

```
iceaa_p.head()
```

	Radios	Unnamed: 1	Unnamed: 2	...	Unnamed: 7	Unnamed: 8	Unnamed: 9
0	Escalation	1.020977	*Using same factor as FY21 --> FY22	...	NaN	NaN	NaN
1	Qty Procured	2024	2025	...	2030	2031	2032
2	NaN	First Award Only (At Award)	13-24 Months After	...	Extrap	Extrap	Extrap
3	1	649699	275373	...	NaN	NaN	NaN
4	2	NaN	NaN	...	NaN	NaN	NaN

5 rows × 10 columns

EXAMPLE (CONT.)

```
iceaa_p = pd.read_excel('Sample_Dataset.xlsx', sheet_name = 'Pricing Tables', skiprows = 3)
```

```
iceaa_p.head()
```

	Unnamed: 0	First Award Only (At Award)	13-24 Months After	...	Extrap.1	Extrap.2	Extrap.3
0	1	649699.0	275373.0	...	NaN	NaN	NaN
1	2	NaN	NaN	...	NaN	NaN	NaN
2	3	NaN	NaN	...	NaN	NaN	NaN
3	4	NaN	NaN	...	NaN	NaN	NaN
4	5	NaN	NaN	...	NaN	NaN	NaN

5 rows × 10 columns

EXAMPLE (CONT.)

```
### The quantity column is not named
### The extrapolation columns are not clearly defined
iceaa_p.rename(columns = {'Unnamed: 0':'quantity', 'Extrap':'Extrap_2029',
                          'Extrap.1':'Extrap_2030', 'Extrap.2':'Extrap_2031',
                          'Extrap.3':'Extrap_2032'}, inplace = True)
```

```
iceaa_p.head()
```

	quantity	First Award Only (At Award)	13-24 Months After	...	Extrap_2030	Extrap_2031	Extrap_2032
0	1	649699.0	275373.0	...	NaN	NaN	NaN
1	2	NaN	NaN	...	NaN	NaN	NaN
2	3	NaN	NaN	...	NaN	NaN	NaN
3	4	NaN	NaN	...	NaN	NaN	NaN
4	5	NaN	NaN	...	NaN	NaN	NaN

5 rows × 10 columns

EXAMPLE (CONT.)

```
iceaa_p.loc[iceaa_p['quantity'] == 1, '49-60 Months After']
```

```
0    273934.0
Name: 49-60 Months After, dtype: float64
```

```
iceaa_p.loc[iceaa_p['quantity'] == 1, '49-60 Months After'].iat[0]
```

```
273934.0
```

```
factor = (iceaa_p.loc[iceaa_p['quantity'] == 1, '49-60 Months After'].iat[0] /
          iceaa_p.loc[iceaa_p['quantity'] == 1, '37-48 Months After'].iat[0])
print(factor)
```

```
1.0220045068572878
```

EXAMPLE (CONT.)

```
### Want to fill in blank rows with proper values -- easiest if we also have
### The Extrap rows started
### .iat[0] means gives the first element

iceaa_p.loc[iceaa_p['quantity'] == 1, 'Extrap_2029'] =  (factor*
                        iceaa_p.loc[iceaa_p['quantity'] == 1, '49-60 Months After'].iat[0])
### Note the wrap in parentheses allows continuation on next line
iceaa_p.loc[iceaa_p['quantity'] == 1, 'Extrap_2030'] =  (factor*
                        iceaa_p.loc[iceaa_p['quantity'] == 1, 'Extrap_2029'].iat[0])
iceaa_p.loc[iceaa_p['quantity'] == 1, 'Extrap_2031'] =  (factor*
                        iceaa_p.loc[iceaa_p['quantity'] == 1, 'Extrap_2030'].iat[0])
iceaa_p.loc[iceaa_p['quantity'] == 1, 'Extrap_2032'] =  (factor*
                        iceaa_p.loc[iceaa_p['quantity'] == 1, 'Extrap_2031'].iat[0])
```

```
cols = iceaa_p.columns[1:]
print(cols)
```

```
Index(['First Award Only (At Award)', '13-24 Months After', '25-36 Months After',
       '37-48 Months After', '49-60 Months After', 'Extrap_2029', 'Extrap_2030', 'Extrap_2031',
       'Extrap_2032'],
      dtype='object')
```

```
print(iceaa_p.loc[iceaa_p.index == 0,cols].values)
```

```
[[649699.         275373.         285074.         268036.
  273934.         279961.78258144 286122.20354604 292418.18153599
  298852.69941679]]
```

EXAMPLE (CONT.)

```
### .values is technically not recommended
### But the alternative might confuse more (but is basically identical)
for i in range(1,60):
    if i < 10:
        iceaa_p.loc[iceaa_p.index == i, cols] = iceaa_p.loc[iceaa_p.index == 0, cols].values
    elif 10 <= i < 20:
        iceaa_p.loc[iceaa_p.index == i, cols] = iceaa_p.loc[iceaa_p.index == 0, cols].values*0.81
    elif 20 <= i < 30:
        iceaa_p.loc[iceaa_p.index == i, cols] = iceaa_p.loc[iceaa_p.index == 10, cols].values*0.81
    elif 30 <= i < 40:
        iceaa_p.loc[iceaa_p.index == i, cols] = iceaa_p.loc[iceaa_p.index == 20, cols].values*0.81
    elif 40 <= i < 50:
        iceaa_p.loc[iceaa_p.index == i, cols] = iceaa_p.loc[iceaa_p.index == 30, cols].values*0.81
    elif i == 50:
        iceaa_p.loc[iceaa_p.index == i, cols] = iceaa_p.loc[iceaa_p.index == 40, cols].values*0.81
    elif 51 <= i < 60:
        iceaa_p.loc[iceaa_p.index == i, cols] = iceaa_p.loc[iceaa_p.index == i-1, cols].values*0.994
```

EXAMPLE (CONT.)

```
iceaa_p.tail()
```

	quantity	First Award Only (At Award)	13-24 Months After	...	Extrap_2030	Extrap_2031	Extrap_2032
54	55	221147.905392	93732.885769	...	97391.755259	99534.812806	101725.027277
55	56	219821.017959	93170.488455	...	96807.404728	98937.603929	101114.677113
56	57	218502.091852	92611.465524	...	96226.560299	98343.978305	100507.989050
57	58	217191.079301	92055.796731	...	95649.200938	97753.914436	99904.941116
58	59	215887.932825	91503.461950	...	95075.305732	97167.390949	99305.511469

5 rows × 10 columns

CONCLUSION

- Python has a tremendous amount of use cases
  - Automating manual/repetitive tasks, creating visuals, automating emails, renaming large batches of files, converting text files to spreadsheets, web scraping, text analysis, etc.
- Anyone can learn and start using Python
- There are lots and lots of resources online
  - e.g. “pandas fill in blanks in column”
- Best way to get started is to pick a task and start working on it