Problem Set 2 (12.5 points)

Data Manipulation in Python

Week 2 References: McKinney, Chapters 5, 6, 7

Getting Started with Pandas - Chapter 5

In [1]:
```
import pandas as pd
```

In [2]:
```
from pandas import Series, DataFrame
```

In [3]:
```
import numpy as np
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
np.set_printoptions(precision=4, suppress=True)
```

Pandas Data Structures

1 Series - show an example of a Series by filling in the missing elements (after the =) in the code below:

In [4]:
```
# Creating a Series with missing elements filled in
obj = {'Laye': 25, 'Saul': 30, 'Faye': None, 'Zack': 28, 'Abdul': None, 'Zee': 34}

obj
```

Out[4]:
```
{'Laye': 25, 'Saul': 30, 'Faye': None, 'Zack': 28, 'Abdul': None, 'Zee': 34}
```

In [5]:
```
# Creating a Series from the dictionary
series_obj = pd.Series(obj)

# Creating a dictionary with fill objects for specific elements
fill_obj = {'Faye': 30, 'Abdul': 22}

# Filling in missing elements with fill objects
series_filled = series_obj.fillna(fill_obj)

print("Original Series:")
print(series_obj)

print("\nSeries with Missing Elements Filled In:")
print(series_filled)
```

```
Original Series:
Laye     25.0
Saul     30.0
Faye      NaN
Zack     28.0
Abdul     NaN
Zee      34.0
dtype: float64

Series with Missing Elements Filled In:
Laye     25.0
Saul     30.0
Faye     30.0
Zack     28.0
Abdul    22.0
Zee      34.0
dtype: float64
```

the DataFrame

2 DataFrame - show an example of a Dataframe by filling in the missing element after the = in the code below:

In [6]:
```
# Creating a DataFrame with multiple missing values

data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada', 'Seattle', 'Seattle'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003, None, 2005],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2,1.8,None]}
frame = pd.DataFrame(data)
```

In [7]:
```
frame
```

Out[7]:

	state	year	pop
0	Ohio	2000.0	1.5
1	Ohio	2001.0	1.7
2	Ohio	2002.0	3.6
3	Nevada	2001.0	2.4
4	Nevada	2002.0	2.9
5	Nevada	2003.0	3.2
6	Seattle	NaN	1.8
7	Seattle	2005.0	NaN

In [8]:
```
# Creating a dictionary with fill values for specific columns

fildata = {
        'year': 2004,
        'pop': 2.6}

# Filling in missing values with specified values for each column
ffill = frame.fillna(fildata)

print("\nDataFrame with Multiple Missing Values Filled In:")
print('\n', ffill)
```

```
DataFrame with Multiple Missing Values Filled In:

      state    year  pop
0      Ohio  2000.0  1.5
1      Ohio  2001.0  1.7
2      Ohio  2002.0  3.6
3    Nevada  2001.0  2.4
4    Nevada  2002.0  2.9
5    Nevada  2003.0  3.2
6   Seattle  2004.0  1.8
7   Seattle  2005.0  2.6
```

3 Display the top 5 rows of the DataFrame just created

In [9]:
```
#Displaying the top 5 rows DataFrame
ffill.head()
```

Out[9]:

	state	year	pop
0	Ohio	2000.0	1.5
1	Ohio	2001.0	1.7
2	Ohio	2002.0	3.6
3	Nevada	2001.0	2.4
4	Nevada	2002.0	2.9

Reindexing

4

In [10]:
```
# Creating a DataFrame using np.arange
frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                     index=['a', 'c', 'd'],
                     columns=['Ohio', 'Texas', 'California'])
frame
```

Out[10]:

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

Given the above DataFrame: Ohio Texas California a 0 1 2 c 3 4 5 d 6 7 8

Make a new DataFrame that includes a new row 'b'.

In [11]:
```
# Adding a new row 'b' to the DataFrame
frame.loc['b'] = [9, 10, 11]

# Displaying the updated DataFrame
print(frame)
```

```
   Ohio  Texas  California
a     0      1           2
c     3      4           5
d     6      7           8
b     9     10          11
```

Dropping Entries from an axis

5 Drop 'c'

In [12]:
```
#Original series and display
obj = pd.Series(np.arange(5.), index=["a", "b", "c", "d", "e"])
obj
```

Out[12]:
```
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
dtype: float64
```

In [13]:
```
# Dropping the element associated with index 'c'
new_obj = obj.drop('c')

#Displaying updated series
new_obj
```

Out[13]:
```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

Indexing Selection and Filtering

6 Display just column "two" from the following DataFrame:

In [14]:
```
# Creating a DataFrame using np.arange
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=["Ohio", "Colorado", "Utah", "New York"],
                    columns=["one", "two", "three", "four"])
data
```

Out[14]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

In [15]:
```
# Displaying just column 'two' from the DataFrame
column_two = data['two']
print(column_two)
```

```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64
```

Selection with loc and iloc

7 Selecting a subset of the rows and columns from a DataFrame using either axis labels (loc) or integers (iloc). Select the Utah row from the 'data' DataFrame and the columns labeled 'four', 'one', and 'two' respectively in that order, first using (loc) and then using (iloc).

In [16]:
```
# Using loc to select the Utah row and specific columns
subset_loc = data.loc['Utah', ['four', 'one', 'two']]
print("Subset using loc:")
print(subset_loc)
```

```
Subset using loc:
four    11
one      8
two      9
Name: Utah, dtype: int64
```

In [17]:
```
# Using iloc to select the Utah row and specific columns by integer positions
subset_iloc = data.iloc[2, [3, 0, 1]]
print("\nSubset using iloc:")
print(subset_iloc)
```

```
Subset using iloc:
four    11
one      8
two      9
Name: Utah, dtype: int64
```

Arithmetic and data alignment

8 Add the 2 DataFrames together to form a DataFrame whose index and columns are the unions of the ones in each DataFrame:

In [19]:
```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list("bcd"),
                   index=["Ohio", "Texas", "Colorado"])
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list("bde"),
                   index=["Utah", "Ohio", "Texas", "Oregon"])
```

In [20]:
```
# Adding the two DataFrames together
result = df1 + df2

# Displaying the result
print("Resulting DataFrame:")
print(result)
```

```
Resulting DataFrame:
            b   c     d   e
Colorado  NaN NaN   NaN NaN
Ohio      3.0 NaN   6.0 NaN
Oregon    NaN NaN   NaN NaN
Texas     9.0 NaN  12.0 NaN
Utah      NaN NaN   NaN NaN
```

Arithmetic methods with fill values

9 Using the add method on df1, pass df2 and 0 as a fill_value.

In [21]:
```
df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
                   columns=list("abcd"))
df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
                   columns=list("abcde"))
df2.loc[1, "b"] = np.nan
```

In [22]:
```
# Using the add method with fill_value
result1 = df1.add(df2, fill_value=0)

# Displaying the result
print("Resulting DataFrame:")
print(result1)
```

```
Resulting DataFrame:
      a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0   5.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

Operations between DataFrame and Series

10 Subtract the Series from the DataFrame

In [23]:
```
# Creating a DataFrame using np.arange
frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                     columns=list("bde"),
                     index=["Utah", "Ohio", "Texas", "Oregon"])
series = frame.iloc[0]
frame
series
```

Out[23]:
```
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64
```

In [24]:
```
# Subtracting the Series from the DataFrame
result2 = frame - series

# Displaying the result
print("Resulting DataFrame:")
print(result2)
```

```
Resulting DataFrame:
          b    d    e
Utah    0.0  0.0  0.0
Ohio    3.0  3.0  3.0
Texas   6.0  6.0  6.0
Oregon  9.0  9.0  9.0
```

Sorting and Ranking

11 Use the data in the 'b' column to sort the DataFrame below:

In [ ]:
```
# Creating DataFrame frame
frame = pd.DataFrame({"b": [4, 7, -3, 2], "a": [0, 1, 0, 1]})
frame
```

In [25]:
```
# Sorting the DataFrame based on the 'b' column in descending order
sorted_frame_descending = frame.sort_values(by='b', ascending=False)

# Displaying the sorted DataFrame in descending order
print("Sorted DataFrame (Descending):")
print(sorted_frame_descending)
```

```
Sorted DataFrame (Descending):
          b     d     e
Oregon  9.0  10.0  11.0
Texas   6.0   7.0   8.0
Ohio    3.0   4.0   5.0
Utah    0.0   1.0   2.0
```

Summarizing and Computing Descriptive Statistics

12 Get summary statistics for the following DataFrame by using one command

In [27]:
```
# Creating DataFrame df
df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
                   [np.nan, np.nan], [0.75, -1.3]],
                  index=["a", "b", "c", "d"],
                  columns=["one", "two"])
df
```

Out[27]:

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

In [28]:
```
# Getting summary statistics for the DataFrame
summary_stats = df.describe()

# Displaying the summary statistics
print("Summary Statistics:")
print(summary_stats)
```

```
Summary Statistics:
            one       two
count  3.000000  2.000000
mean   3.083333 -2.900000
std    3.493685  2.262742
min    0.750000 -4.500000
25%    1.075000 -3.700000
50%    1.400000 -2.900000
75%    4.250000 -2.100000
max    7.100000 -1.300000
```

Correlation and Covariance

14

In [29]:
```
#Installing pandas data reader library
%pip install pandas-datareader
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas-datareader in /usr/local/lib/python3.6/site-packages (0.10.0)
Requirement already satisfied: requests>=2.19.0 in /usr/lib/python3.6/site-packages (from pandas-datareader) (2.21.0)
Requirement already satisfied: pandas>=0.23 in /usr/local/lib64/python3.6/site-packages (from pandas-datareader) (1.0.3)
Requirement already satisfied: lxml in /usr/local/lib64/python3.6/site-packages (from pandas-datareader) (4.9.1)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib64/python3.6/site-packages (from pandas>=0.23->pandas-datareader) (1.19.5)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/site-packages (from pandas>=0.23->pandas-datareader) (2019.3)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/lib/python3.6/site-packages (from pandas>=0.23->pandas-datareader) (2.8.0)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/site-packages (from requests>=2.19.0->pandas-datareader) (2.8)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/lib/python3.6/site-packages (from requests>=2.19.0->pandas-datareader) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3.6/site-packages (from requests>=2.19.0->pandas-datareader) (2019.3.9)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/site-packages (from requests>=2.19.0->pandas-datareader) (1.24.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/site-packages (from python-dateutil>=2.6.1->pandas>=0.23->pandas-datareader) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

Restart the Kernel

In [1]:
```
#Importing pandas library with an alias
import pandas as pd
```

In [2]:
```
#Importing series and dataframe from pandas library
from pandas import Series, DataFrame
```

In [3]:
```
import numpy as np
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
np.set_printoptions(precision=4, suppress=False)
```

In [4]:
```
import pandas_datareader.data as web
```

In [6]:
```
price = pd.read_pickle("yahoo_price.pkl")
volume = pd.read_pickle("yahoo_volume.pkl")
```

In [7]:
```
returns = price.pct_change()
returns.tail()
```

Out[7]:

Date	AAPL	GOOG	IBM	MSFT
2016-10-17	-0.000680	0.001837	0.002072	-0.003483
2016-10-18	-0.000681	0.019616	-0.026168	0.007690
2016-10-19	-0.002979	0.007846	0.003583	-0.002255
2016-10-20	-0.000512	-0.005652	0.001719	-0.004867
2016-10-21	-0.003930	0.003011	-0.012474	0.042096

Find the Correlation in returns between MSFT and IBM:

In [8]:
```
# Correlation in returns between MSFT and IBM
correlation_msft_ibm = returns['MSFT'].corr(returns['IBM'])

# Displaying the correlation
print("Correlation between MSFT and IBM returns:")
print(correlation_msft_ibm)
```

```
Correlation between MSFT and IBM returns:
0.49976361144151144
```

Find the Covariance in returns betwen MSFT and IBM:

In [9]:
```python
# Covariance in returns between MSFT and IBM
covariance_msft_ibm = returns['MSFT'].cov(returns['IBM'])

# Displaying the covariance
print("Covariance between MSFT and IBM returns:")
print(covariance_msft_ibm)
```

```
Covariance between MSFT and IBM returns:
8.870655479703546e-05
```

Unique Values, Value Counts, and Membership

15 Count the "answers" to each "question" in the DataFrame below, filling in 0 for missing:

In [11]:
```python
data = pd.DataFrame({"Qu1": [1, 3, 4, 3, 4],
                     "Qu2": [2, 3, 1, 2, 3],
                     "Qu3": [1, 5, 2, 4, 4]})
data
```

Out[11]:

	Qu1	Qu2	Qu3
0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4

In [12]:
```python
# Counting the answers for each question and filling in 0 for missing values
result = data.apply(lambda x: x.value_counts()).fillna(0)

# Displaying the result
result
```

Out[12]:

	Qu1	Qu2	Qu3
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

Data Loading, Storage and File Formats - Chapter 6

In [13]:
```python
import numpy as np
import pandas as pd
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4, suppress=True)
```

Reading and Writing Data in Text Format

16 Read (and display) a CSV file (upload only CSV UTF-8 files to JupyterHub) into a pandas DataFrame

In [15]:
```python
# define the CSV file
file_path = 'macrodata.csv'

# Reading the CSV file into a Pandas DataFrame
df18 = pd.read_csv(file_path)

# Displaying the DataFrame
df18
```

Out[15]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp	pop	infl	realint
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.980	139.7	2.82	5.8	177.146	0.00	0.00
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.150	141.7	3.08	5.1	177.830	2.34	0.74
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.350	140.5	3.82	5.3	178.657	2.74	1.09
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.370	140.0	4.33	5.6	179.386	0.27	4.06
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.540	139.6	3.50	5.2	180.007	2.31	1.19
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
198	2008.0	3.0	13324.600	9267.7	1990.693	991.551	9838.3	216.889	1474.7	1.17	6.0	305.270	-3.16	4.33
199	2008.0	4.0	13141.920	9195.3	1857.661	1007.273	9920.4	212.174	1576.5	0.12	6.9	305.952	-8.79	8.91
200	2009.0	1.0	12925.410	9209.2	1558.494	996.287	9926.4	212.671	1592.8	0.22	8.1	306.547	0.94	-0.71
201	2009.0	2.0	12901.504	9189.0	1456.678	1023.528	10077.5	214.469	1653.6	0.18	9.2	307.226	3.37	-3.19
202	2009.0	3.0	12990.341	9256.0	1486.398	1044.088	10040.6	216.385	1673.9	0.12	9.6	308.013	3.56	-3.44

203 rows × 14 columns

Reading Text Files in Pieces

17 Make the pandas display settings more compact by displaying a maximum of 10 rows:

In [16]:
```python
# Setting display options for maximum rows
pd.set_option('display.max_rows', 10)

# Displaying the DataFrame
df18
```

Out[16]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp	pop	infl	realint
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.980	139.7	2.82	5.8	177.146	0.00	0.00
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.150	141.7	3.08	5.1	177.830	2.34	0.74
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.350	140.5	3.82	5.3	178.657	2.74	1.09
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.370	140.0	4.33	5.6	179.386	0.27	4.06
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.540	139.6	3.50	5.2	180.007	2.31	1.19
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
198	2008.0	3.0	13324.600	9267.7	1990.693	991.551	9838.3	216.889	1474.7	1.17	6.0	305.270	-3.16	4.33
199	2008.0	4.0	13141.920	9195.3	1857.661	1007.273	9920.4	212.174	1576.5	0.12	6.9	305.952	-8.79	8.91
200	2009.0	1.0	12925.410	9209.2	1558.494	996.287	9926.4	212.671	1592.8	0.22	8.1	306.547	0.94	-0.71
201	2009.0	2.0	12901.504	9189.0	1456.678	1023.528	10077.5	214.469	1653.6	0.18	9.2	307.226	3.37	-3.19
202	2009.0	3.0	12990.341	9256.0	1486.398	1044.088	10040.6	216.385	1673.9	0.12	9.6	308.013	3.56	-3.44

203 rows × 14 columns

18 Read a CSV file but only read a small number of rows:

In [17]:
```python
# Reading a small number of rows (e.g., 5) from the CSV file into a Pandas DataFrame
df18 = pd.read_csv(file_path, nrows=5)

# Displaying the DataFrame
df18
```

Out[17]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp	pop	infl	realint
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8	177.146	0.00	0.00
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1	177.830	2.34	0.74
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3	178.657	2.74	1.09
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6	179.386	0.27	4.06
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2	180.007	2.31	1.19

Reading Microsoft Excel Files

20 Read an Excel file of type *.xlsx* in 2 ways, ExcelFile/read_excel and just using read_excel

In [18]:
```python
%pip install xlrd # allows read of xls files
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: xlrd in /usr/local/lib/python3.6/site-packages (2.0.1)
Note: you may need to restart the kernel to use updated packages.
```

In [19]:
```python
%pip install openpyxl # allows read of xlsx files
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: openpyxl in /usr/local/lib/python3.6/site-packages (3.0.10)
Requirement already satisfied: et-xmlfile in /usr/local/lib/python3.6/site-packages (from openpyxl) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

Restart the Kernel

In [3]:
```
import pandas as pd
```

In [7]:
```
# ExcelFile/read_excel
xlsx = pd.ExcelFile('boston.xlsx', engine='openpyxl')
```

In [8]:
```
# Reading Data sheet into a DataFrame

pd.read_excel(xlsx, 'Data')
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [9]:
```
#  Using read_excel
frame = pd.read_excel(
    'boston.xlsx', 'Data', engine='openpyxl')
frame
```

Out[9]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

Data Cleaning and Preparation - Chapter 7

"During the course of doing data analyisis and modeling, a significant amount of time is spent on data preparation: loading, cleaning, transforming, and rearranging. Such task are often reported to take up [to] 80% or more of an analytst's time" - McKinney

In [3]:
```
import numpy as np
import pandas as pd
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4, suppress=False)
```

Handling Missing Data

21 Return boolean values indicationg which values are missing/NA

In [4]:
```
string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
string_data
```

Out[4]:
```
0     aardvark
1    artichoke
2          NaN
3      avocado
dtype: object
```

In [5]:
```
# Checking for missing/NA values using isna() or isnull()
missing_values_mask = string_data.isna()

# Displaying the boolean mask
print("Boolean mask for missing/NA values:")
print(missing_values_mask)
```

```
Boolean mask for missing/NA values:
0    False
1    False
2     True
3    False
dtype: bool
```

Filtering out Missing Data

22 Drop any row containing a missing value

In [6]:
```
data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
                     [NA, NA, NA], [NA, 6.5, 3.]])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-afa12c79a244> in <module>
----> 1 data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],
      2                      [NA, NA, NA], [NA, 6.5, 3.]])

NameError: name 'NA' is not defined
```

In [9]:
```
# Creating the correct Pandas DataFrame named data
cleaned_data = pd.DataFrame([[1., 6.5, 3.], [1., np.nan, np.nan],
                     [np.nan, np.nan, np.nan], [np.nan, 6.5, 3.]])

# Dropping rows containing missing values
cleaned = cleaned_data.dropna()

# Displaying the DataFrame without missing values
print("DataFrame after dropping rows with missing values:")
print(cleaned)
```

```
DataFrame after dropping rows with missing values:
     0    1    2
0  1.0  6.5  3.0
```

Drop only those rows that are all NA:

In [10]:
```
# Dropping rows where all values are NA
data_without_all_missing = cleaned_data.dropna(how='all')

# Displaying the DataFrame without rows where all values are missing
print("DataFrame after dropping rows with all missing values:")
print(data_without_all_missing)
```

```
DataFrame after dropping rows with all missing values:
     0    1    2
0  1.0  6.5  3.0
1  1.0  NaN  NaN
3  NaN  6.5  3.0
```

Filling in Missing Data

23 Replace all missing data with zeroes

In [11]:
```
df23 = pd.DataFrame(np.random.randn(7, 3))
df23.iloc[:4, 1] = np.nan
df23.iloc[:2, 2] = np.nan
df23
```

Out[11]:

	0	1	2
0	-0.204708	NaN	NaN
1	-0.555730	NaN	NaN
2	0.092908	NaN	0.769023
3	1.246435	NaN	-1.296221
4	0.274992	0.228913	1.352917
5	0.886429	-2.001637	-0.371843
6	1.669025	-0.438570	-0.539741

In [12]:
```
# Replacing missing values with zeroes
df_filled_zeroes = df23.fillna(0)

# Displaying the DataFrame with missing values replaced by zeroes
print("DataFrame after replacing missing values with zeroes:")
print(df_filled_zeroes)
```

```
DataFrame after replacing missing values with zeroes:
          0         1         2
0 -0.204708  0.000000  0.000000
1 -0.555730  0.000000  0.000000
2  0.092908  0.000000  0.769023
3  1.246435  0.000000 -1.296221
4  0.274992  0.228913  1.352917
5  0.886429 -2.001637 -0.371843
6  1.669025 -0.438570 -0.539741
```

Data Transformation

Removing Duplicates

24 Use the method that returns a Boolean Series indicating whether each row is a duplicate

In [13]:
```
#pandas data frame
data24 = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                      'k2': [1, 1, 2, 3, 3, 4, 4]})
data24
```

Out[13]:

	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4

In [14]:
```
# Creating a boolean Series indicating whether each row is a duplicate
is_duplicate_series = data24.duplicated()

# Displaying the boolean Series
print("Boolean Series indicating whether each row is a duplicate:")
print(is_duplicate_series)
```

```
Boolean Series indicating whether each row is a duplicate:
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

Use the method that removes those duplicate rows

In [15]:
```
# Removing duplicate rows
data_no_duplicates = data24.drop_duplicates()

# Displaying the DataFrame without duplicates
print("DataFrame after removing duplicate rows:")
print(data_no_duplicates)
```

```
DataFrame after removing duplicate rows:
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
```

Transforming Data using a Function or Mapping

25

In [33]:
```
# the raw data
data25 = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
                                'Pastrami', 'corned beef', 'Bacon',
                                'pastrami', 'honey ham', 'nova lox'],
                      'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
```

In [34]:
```
# the mapping:
meat_to_animal = {
  'bacon': 'pig',
  'pulled pork': 'pig',
  'pastrami': 'cow',
  'corned beef': 'cow',
  'honey ham': 'pig',
  'nova lox': 'salmon'
}
```

Convert each value of food to lowercase

In [35]:
```
# Converting each value in the 'food' column to lowercase
data25['food'] = data25['food'].str.lower()

# Displaying the updated DataFrame
data25
```

Out[35]:

	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	pastrami	6.0
4	corned beef	7.5
5	bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

create a new column 'animal' containing the mapped meat:

In [37]:
```
# Creating a new column 'animal' containing the mapped meat values
data25['animal'] = data25['food'].map(meat_to_animal)

# Displaying the updated DataFrame
data25
```

Out[37]:

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	pastrami	6.0	cow
4	corned beef	7.5	cow
5	bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

Replacing Values

26 In one line of code, replace the -999 in the following Series with NA values and the -1000 with 0.

In [40]:
```
# Original data
data26 = pd.Series([1., -999., 2., -999., -1000., 3.])

#Displaying Original data
data26
```

Out[40]:
```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

In [41]:
```
# mapping to relpacements
data26 = data26.replace({-999.: np.nan, -1000.: 0})

#Displaying replacements
data26
```

Out[41]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

Renaming Axis Indexes

27 Rename both the index OHIO to INDIANA and the column three to peekaboo and modify the DataFrame below in-place:

In [46]:
```
# Creating a Pandas DataFrame named data27
data27 = pd.DataFrame(np.arange(12).reshape((3, 4)),
                    index=['Ohio', 'Colorado', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
#Displaying the original data27
data27
```

Out[46]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
New York	8	9	10	11

In [47]:
```
# Function to transform index values to uppercase
transform = lambda x: x[:4].upper()

#displaying index values and data types
data27.index.map(transform)
```

Out[47]:
```
Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

In [48]:
```
#make sure the data27 index is store
data27.index = data27.index.map(transform)

#display updated dataframe
data27
```

Out[48]:

	one	two	three	four
OHIO	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

In [49]:
```
# Renaming index and column in-place
data27.rename(index={'Ohio': 'INDIANA'}, columns={'three': 'peekaboo'}, inplace=True)
```

In [50]:
```
# Displaying the modified DataFrame
print("Modified DataFrame:")
print(data27)
```

```
Modified DataFrame:
      one  two  peekaboo  four
OHIO    0    1         2     3
COLO    4    5         6     7
NEW     8    9        10    11
```

Discretization and Binning

28

In [58]:
```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

In [59]:
```
bins = [18, 25, 35, 60, 100]
```

Produce the following output:

[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]] Length: 12 Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]

In [63]:
```
# Create a pandas DataFrame
df = pd.DataFrame({'Age': ages})

# Bin the ages into the specified ranges
df['Category'] = pd.cut(df['Age'], bins=bins)

# Output the requested format
output = list(df['Category'])
length = len(output)
categories = df['Category'].cat.categories

output_int = f"{output} Length: {length} Categories {categories}"

print(output_int)
```

```
[Interval(18, 25, closed='right'), Interval(18, 25, closed='right'), Interval(18, 25, closed='right'), Interval(25, 35, closed='right'), Interval(18, 25, closed='right'), Interval(18, 25, closed='right'), Interval(35, 60, closed='right'), Interval(2
5, 35, closed='right'), Interval(60, 100, closed='right'), Interval(35, 60, closed='right'), Interval(35, 60, closed='right'), Interval(25, 35, closed='right')] Length: 12 Categories IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]],
              closed='right',
              dtype='interval[int64]')
```

Produce the following output: array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)

In [64]:
```
# Provided data
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]

# Bin the ages into the specified ranges
categories = np.digitize(ages, bins=bins, right=True)

# Output the requested array
output_array = categories - 1  # Adjusting to start from 0
output_array = output_array.astype(np.int8)

print(output_array)
```

```
[0 0 0 1 0 0 2 1 3 2 2 1]
```

Produce the following output:

IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]], closed='right', dtype='interval[int64]')

In [66]:
```
# Provided data
bins = [18, 25, 35, 60, 100]

# Create the IntervalIndex
interval_index = pd.IntervalIndex.from_breaks(bins, closed='right')

print(interval_index)
```

```
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]],
              closed='right',
              dtype='interval[int64]')
```

Produce the following output:

(18, 25] 5

(35, 60] 3

(25, 35] 3

(60, 100] 1

dtype: int64

In [67]:
```
# Provided data
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]

# Create a pandas DataFrame
df = pd.DataFrame({'Age': ages})

# Bin the ages into the specified ranges
df['Category'] = pd.cut(df['Age'], bins=bins)

# Count occurrences of each interval
count_by_interval = df['Category'].value_counts().sort_index()

print(count_by_interval)
```

```
(18, 25]     5
(25, 35]     3
(35, 60]     3
(60, 100]    1
Name: Category, dtype: int64
```

Produce the following output:

[Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAged, MiddleAged, YoungAdult]

Length: 12

Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]

In [68]:
```
# Provided data
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
labels = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']

# Bin the ages into the specified ranges and assign custom labels
categories = pd.cut(ages, bins=bins, labels=labels, right=True)

# Output the requested format
output_list = list(categories.astype(str))
length = len(output_list)

output_str = f"{output_list} Length: {length}"

print(output_str)
```

```
['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', 'Youth', 'MiddleAged', 'YoungAdult', 'Senior', 'MiddleAged', 'MiddleAged', 'YoungAdult'] Length: 12
```

In [69]:
```
# Bin the ages into the specified ranges and assign custom labels
categories = pd.cut(ages, bins=bins, labels=labels, right=True)

# Specify the order of the custom labels
custom_order = pd.CategoricalDtype(categories=labels, ordered=True)
categories = categories.astype(custom_order)

# Output the requested format
output_str = f"Categories {categories.categories}"

print(output_str)
```

```
Categories Index(['Youth', 'YoungAdult', 'MiddleAged', 'Senior'], dtype='object')
```

Detecting and Filtering Outliers

29 Select all rows having a value exceeding 3 or -3

In [54]:
```
# Creating a Pandas DataFrame named data
data29 = pd.DataFrame(np.random.randn(1000, 4))
#print summary statistics
data29.describe()
```

Out[54]:

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.000562	-0.017333	-0.024793	-0.018473
std	1.024157	0.994945	0.961976	0.980492
min	-3.645860	-3.481593	-3.194414	-3.108915
25%	-0.697678	-0.694020	-0.701202	-0.695115
50%	0.033173	-0.009461	-0.035084	0.005374
75%	0.686028	0.653045	0.650671	0.618965
max	3.189940	3.525865	3.023720	2.859053

In [71]:
```
# Select rows where any value exceeds 3 or is less than -3
selected_rows = data29[(data29 > 3) | (data29 < -3).any(axis=0)]


# Displaying the selected rows
print("Selected rows where any value exceeds 3 or falls below -3:")
print(selected_rows)
```

```
Selected rows where any value exceeds 3 or falls below -3:
            0         1         2         3
0    0.215523 -2.056737 -1.248733  1.266970
1    0.722045 -0.954567  0.943233  1.192702
2    1.035828  1.031435  0.179642 -0.625160
3    1.754117  0.665097  0.996054  1.254051
4   -0.071556  1.140204 -0.139397  0.130148
..        ...       ...       ...       ...
995 -0.075264  0.112345  0.166874  0.012628
996  0.815313 -0.732001  0.868791  0.149693
997  0.485218  0.161056 -1.068808  1.190359
998 -1.053204  0.776001  1.311260  1.159677
999  0.477395 -0.004493  0.574631  1.094319

[1000 rows x 4 columns]
```

Permutation and Random Sampling

30 Randomly reorder the rows in the following DataFrame

In [73]:
```
#original dataframe
df30 = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
```

In [74]:
```
# Randomly reorder the rows
sampler = df30.sample(frac=1, random_state=42)
sampler
```

Out[74]:

	0	1	2	3
1	4	5	6	7
4	16	17	18	19
2	8	9	10	11
0	0	1	2	3
3	12	13	14	15

Use a function to use the new array

In [83]:
```
#Create a function
def randomize_rows(sampler, random_state=None):
    randomized_df = sampler.sample(frac=1, random_state=random_state)
    return randomized_df
```

In [84]:
```
# Assuming you have already created the DataFrame named df30
df30_randomized = randomize_rows(sampler, random_state=42)

print(df30_randomized)
```

```
    0   1   2   3
4  16  17  18  19
3  12  13  14  15
2   8   9  10  11
1   4   5   6   7
0   0   1   2   3
```

Generate a random subset with replacement

In [85]:
```
#Create a series
choices = pd.Series([5, 7, -1, 6, 4])
```

In [86]:
```
# Set seed for reproducibility
np.random.seed(42)

# Generate a random subset with replacement
draws = np.random.choice(choices, size=len(choices), replace=True)
```

In [87]:
```
#Display the draws
draws
```

Out[87]: `array([ 6,  4, -1,  4,  4])`

Computing Indicator/Dummy Variables

converting a categorical variable into a "dummy" or "indicator" matrix

31

a b c

0 0 1 0 1 0 1 0 2 1 0 0 3 0 0 1 4 1 0 0 5 0 1 0

In [89]:
```
#Original dataframe
df31 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                     'data1': range(6)})
```

In [90]:
```
# Convert 'key' column into dummy variables
dummy_df = pd.get_dummies(df31['key'], prefix='key')

# Concatenate the dummy variables with the original DataFrame
df31_with_dummies = pd.concat([df31, dummy_df], axis=1)

print(df31_with_dummies)
```

```
  key  data1  key_a  key_b  key_c
0   b      0      0      1      0
1   b      1      0      1      0
2   a      2      1      0      0
3   c      3      0      0      1
4   a      4      1      0      0
5   b      5      0      1      0
```

String Manipulation

String Object Methods

32

In [92]:
```
#String object
val = 'a,b,  guido'
```

break the comma-separated string into pieces

In [93]:
```
# Split the string into pieces
pieces = val.split(',')

# Remove leading and trailing whitespaces from each piece
cleaned_pieces = [piece.strip() for piece in pieces]

print(cleaned_pieces)
```

```
['a', 'b', 'guido']
```

trim the withespace from the resulting elements

In [94]:
```
# Split the string into pieces and remove leading/trailing whitespaces
cleaned_pieces = [piece.strip() for piece in val.split(',')]

print(cleaned_pieces)
```

```
['a', 'b', 'guido']
```

Regular Expressions

33 Split the string with a variable number of whitespace characters using regex

In [96]:
```
import re
text = "foo    bar\t baz  \tqux"
```

In [97]:
```
# Split the string with a variable number of whitespace characters
split_result = re.split(r'\s+', text)

print(split_result)
```

```
['foo', 'bar', 'baz', 'qux']
```

34 Use a method to list out the email addresses:

In [102]:
```
text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com
"""
```

In [103]:
```
pattern = re.findall(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', text)

# Print the list of email addresses
print(pattern)
```

```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

In [104]:
```
# re.IGNORECASE makes the regex case-insensitive
regex = re.findall(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b', text, flags=re.IGNORECASE)

# Print the list of case-insensitive email addresses
print(regex)
```

```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

35 Find email addresses and simultaneously segment each address into its 3 components: username, domain name, and domain suffix.

In [105]:
```
# Extract email addresses and components using re.findall() with re.IGNORECASE
email_info = re.findall(r'\b([A-Za-z0-9._%+-]+)@([A-Za-z0-9.-]+)\.([A-Z|a-z]{2,})\b', text, flags=re.IGNORECASE)

# Print the list of email information
for username, domain_name, domain_suffix in email_info:
    print(f"Username: {username}, Domain: {domain_name}, Suffix: {domain_suffix}")
```

```
Username: dave, Domain: google, Suffix: com
Username: steve, Domain: gmail, Suffix: com
Username: rob, Domain: gmail, Suffix: com
Username: ryan, Domain: yahoo, Suffix: com
```

In [117]:
```
# Given email address
email_address = 'wesm@bright.net'

# Extract email components using re.findall() with re.IGNORECASE
email_info = re.findall(r'\b([A-Za-z0-9._%+-]+)@([A-Za-z0-9.-]+)\.([A-Z|a-z]{2,})\b', email_address, flags=re.IGNORECASE)

# Print the email information
if email_info:
    username, domain_name, domain_suffix = email_info[0]
    print(f'{username, domain_name, domain_suffix}')
else:
    print("Invalid email address format.")
```

```
('wesm', 'bright', 'net')
```

Vectorized String Functions in pandas

36 find the missing data in the following column

In [119]:
```
data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
        'Rob': 'rob@gmail.com', 'Wes': np.nan}
data = pd.Series(data)
# Check for missing data in the 'Wes' column
missing_data = data.isna()

# Print the result
print(missing_data)
```

```
Dave     False
Steve    False
Rob      False
Wes       True
dtype: bool
```

Use the array-oriented method in Series for string operations that skip NA values. Check whether each email address has 'gmail' in it:

```
In [120]: # Check if each email address contains 'gmail'
contains_gmail = data.str.contains('gmail')

# Print the result
print(contains_gmail)
```

```
Dave     False
Steve     True
Rob       True
Wes        NaN
dtype: object
```

Separate out the elements by using regular expressions:

```
In [121]: # Define a regex pattern to capture components
pattern = r'(?P<Username>[A-Za-z0-9._%+-]+)@(?P<Domain>[A-Za-z0-9.-]+)\.(?P<Suffix>[A-Z|a-z]{2,})'

# Use str.extract with regex pattern
extracted_data = data.str.extract(pattern, flags=re.IGNORECASE)

# Print the extracted data
print(extracted_data)
```

```
      Username  Domain Suffix
Dave      dave  google    com
Steve    steve   gmail    com
Rob        rob   gmail    com
Wes        NaN     NaN    NaN
```

Separate out the elements by using vectorized element retrieval:

```
In [122]: # Use str.split to split the email addresses into parts and convert to tuples
split_data = data.str.split('@')
tuple_data = split_data.apply(lambda x: tuple(x) if isinstance(x, list) else np.nan)

# Print the result
print(tuple_data)
```

```
Dave     (dave, google.com)
Steve    (steve, gmail.com)
Rob        (rob, gmail.com)
Wes                     NaN
dtype: object
```

```
In [123]: # Extract domain names using str.extract and regex
domain_names = data.str.extract(r'@([A-Za-z0-9.-]+)\.[A-Z|a-z]{2,}', flags=re.IGNORECASE)

# Print the result
print(domain_names)
```

```
            0
Dave   google
Steve   gmail
Rob     gmail
Wes       NaN
```

Use an alternative way to separate out the elements by using vectorized element retrieval:

```
In [124]: # Extract the part before '@' and the first character of the domain
usernames = data.str.split('@').str[0]
first_char_domain = data.str.extract(r'@([A-Za-z0-9])', expand=False)

# Create a new Series with the extracted elements
result_series = usernames + '@' + first_char_domain

# Print the result
print(result_series)
```

```
Dave      dave@g
Steve    steve@g
Rob        rob@g
Wes          NaN
dtype: object
```

Return the captured groups of a regular expression as a DataFrame: 0 1 2 Dave dave google com Steve steve gmail com Rob rob gmail com Wes NaN NaN NaN

```
In [125]: # Define a regex pattern to capture components
pattern = r'(?P<Username>[A-Za-z0-9._%+-]+)@(?P<Domain>[A-Za-z0-9.-]+)\.(?P<Suffix>[A-Z|a-z]{2,})'

# Use str.extract with regex pattern and convert to DataFrame
captured_groups_df = data.str.extract(pattern, flags=re.IGNORECASE)

# Print the result
print(captured_groups_df)
```

```
      Username  Domain Suffix
Dave      dave  google    com
Steve    steve   gmail    com
Rob        rob   gmail    com
Wes        NaN     NaN    NaN
```

```
In [ ]:
```