Practical Data Science
Assignment 4

Magon Bowling

February 20, 2021

1 Section: Concepts

1. Why do we want to import packages?
   Packages are a collection of functions and data sets developed by the programming community. They make it possible to access data sets, use specific features and functions, and code according to specified desires.

2. What is the use of the "as" code when importing Python packages?
   When you import a package in Python, say pandas or numpy, the "as" command is used to reference the package in code later. This is a shorter notation that links the code to specific functions and programming.

3. How do we save output generated by Python code?
   Set the output "=" to a variable name.

4. How do we save output generated by R code?
   Store the output as a variable using "$<-$".

5. Why would we want to save output?
   To use it later in any calculations.

6. How do we get a data set into Python?
   We import the data first and then perform a "read.csv" command.

7. Why is it important to specify if our data set as column headings or not?
   Rows and columns are listed in a specific order, rows first. Having a specific list of the columns is necessary to reference and use throughout a code.

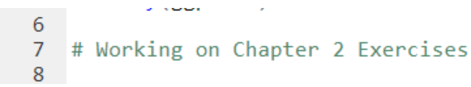
8. What are the two ways we can get a data set into R?
   Data sets can be manually uploaded into R and then use read.csv of the file stored as a data set. Another way to get a data set into R is to read.csv the pathway to the data set, such as a url.

2 Section: Working with the Data

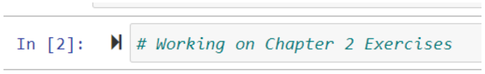
9. Download the program and open the compiler (both R and Python). What is contained in the bottom-right window? The left(for Python) or top-left(for R)?
   In both R and Python we have various windows. There is a console to show the results of

your work, there is the coding window to write and save code, and there is other window sections for files, help support, and more.

10. Type a comment stating that you are working on Chapter 2 exercises.
    In both R and Python, comments are made with ”#” before the text like so:



```
# Working on Chapter 2 Exercises
```



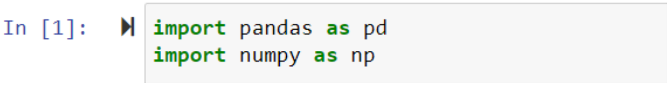
```
In [2]:  # Working on Chapter 2 Exercises
```

11. Locate the “Run” button and note whether there is a keyboard shortcut.
    In both R and Python programming, I use the shortcut of ”ctrl+enter” to run my code. As long as the cursor is on the line of code I want to run, then when I select ”ctrl+enter” the line will run. If I want to run more lines of code, then I select all the lines I want to run and then hit ”ctrl+enter.”

12. Execute the comment from the previous exercise. What is the output? Explain your answer.
    When you execute a comment there is no output. The program recognizes the text as just that and it will not run any code.

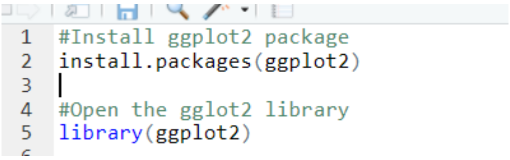
13. Import the following packages:

    (1) For Python, import the *pandas* and *numpy*. Rename the *pandas* package “pd” and rename the *numpy* package “np”.



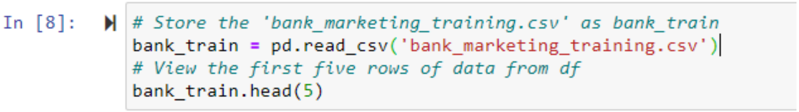
```
In [1]:  import pandas as pd
         import numpy as np
```

    (2) For R, import the *ggplot2*. Make sure you both install and open the package.

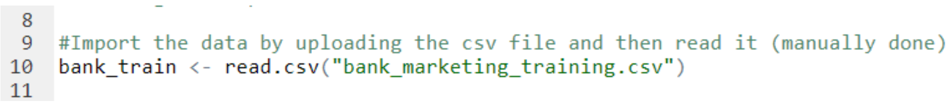


```
#Install ggplot2 package
install.packages(ggplot2)

#Open the gglot2 library
library(ggplot2)
```

14. Import the ”bank_marketing_training” data set and name it ”bank_train.”

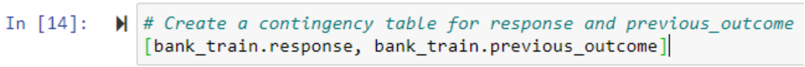


```
In [8]:  # Store the 'bank_marketing_training.csv' as bank_train
         bank_train = pd.read_csv('bank_marketing_training.csv')
         # View the first five rows of data from df
         bank_train.head(5)
```

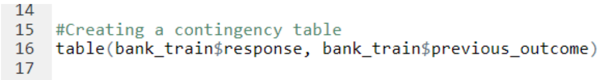


```
#Import the data by uploading the csv file and then read it (manually done)
bank_train <- read.csv("bank_marketing_training.csv")
```

15. Create a contingency table of the variables *response* and *previous_outcome* from the ”bank_train” data set. Do not save the output from the code.

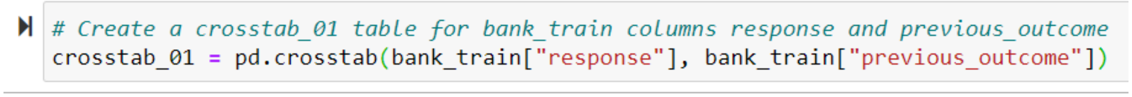


```
In [14]:   # Create a contingency table for response and previous_outcome
           [bank_train.response, bank_train.previous_outcome]
```

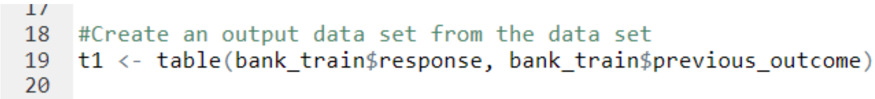


```

#Creating a contingency table
table(bank_train$response, bank_train$previous_outcome)

```

16. Rerun the code from the previous exercise, this time saving the output as ”crosstab_01” (for Python code) or ”t1” (for R code).

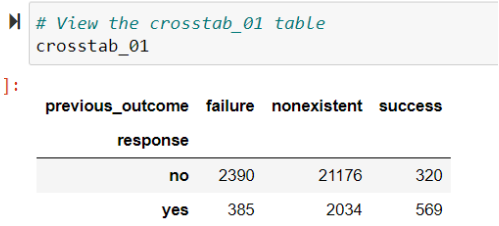


```
# Create a crosstab_01 table for bank_train columns response and previous_outcome
crosstab_01 = pd.crosstab(bank_train["response"], bank_train["previous_outcome"])
```



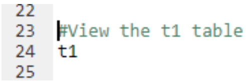
```
#Create an output data set from the data set
t1 <- table(bank_train$response, bank_train$previous_outcome)

```

17. After saving the output in the previous exercise, display the output using the name of the saved output.



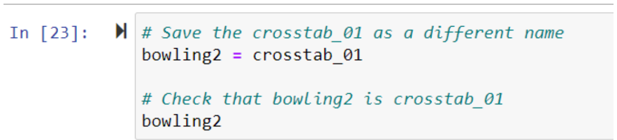
```
# View the crosstab_01 table
crosstab_01
```

```
]:
   previous_outcome  failure  nonexistent  success
           response
                 no     2390        21176      320
                yes      385         2034      569
```

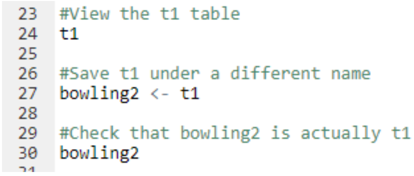


```

#View the t1 table
t1

```

18. Save the contingency table under a different name. This time, use your last name and favorite number as the name; for example *thomas4*.

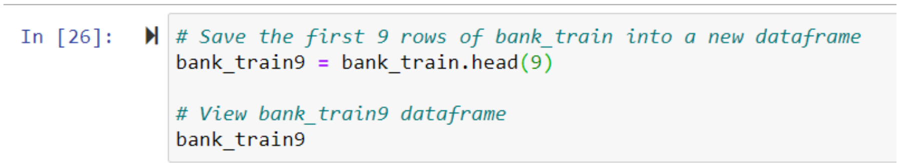


```
In [23]:   # Save the crosstab_01 as a different name
           bowling2 = crosstab_01

           # Check that bowling2 is crosstab_01
           bowling2
```

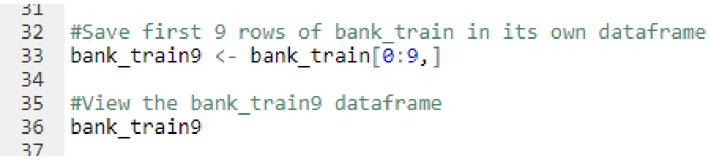


```
#View the t1 table
t1

#Save t1 under a different name
bowling2 <- t1

#Check that bowling2 is actually t1
bowling2
```

19. Save the first nine records of the "bank_train" data set as their own data frame.

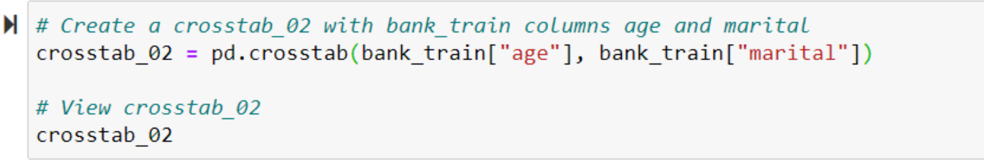


```
# Save the first 9 rows of bank_train into a new dataframe
bank_train9 = bank_train.head(9)

# View bank_train9 dataframe
bank_train9
```

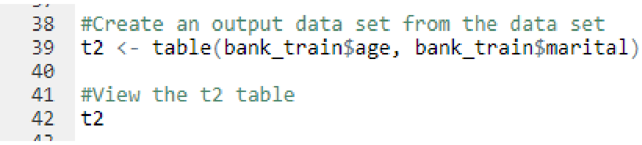


```
#Save first 9 rows of bank_train in its own dataframe
bank_train9 <- bank_train[0:9,]

#View the bank_train9 dataframe
bank_train9
```

20. Save the *age* and *marital* records of the "bank_train" data set as their own data frame.

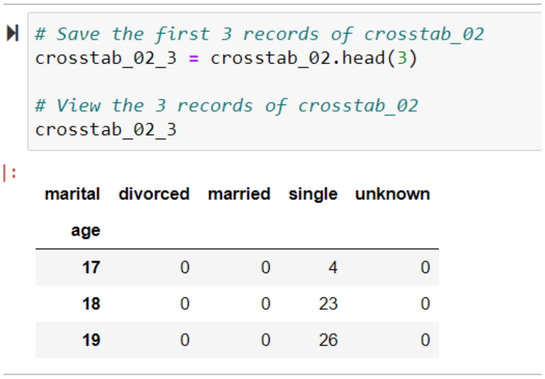


```
# Create a crosstab_02 with bank_train columns age and marital
crosstab_02 = pd.crosstab(bank_train["age"], bank_train["marital"])

# View crosstab_02
crosstab_02
```



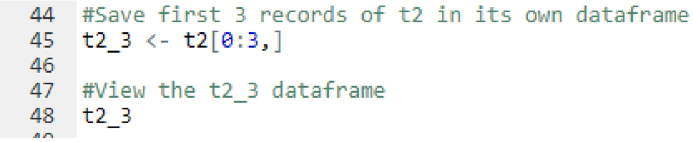
```
#Create an output data set from the data set
t2 <- table(bank_train$age, bank_train$marital)

#View the t2 table
t2
```

21. Save the first three records of the *age* and *marital* variables as their own data frame.



```
# Save the first 3 records of crosstab_02
crosstab_02_3 = crosstab_02.head(3)

# View the 3 records of crosstab_02
crosstab_02_3
```

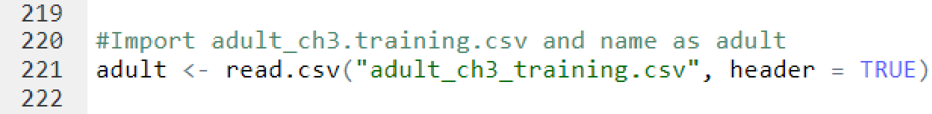
marital	divorced	married	single	unknown
age				
17	0	0	4	0
18	0	0	23	0
19	0	0	26	0



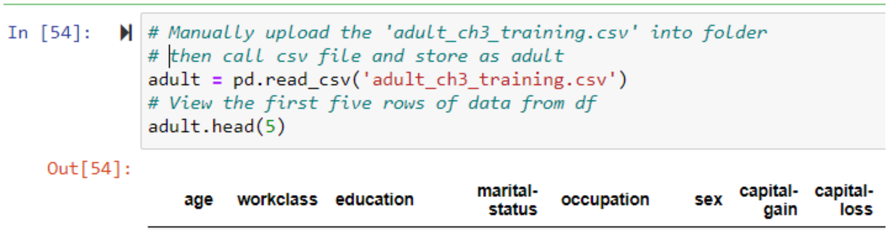
```
#Save first 3 records of t2 in its own dataframe
t2_3 <- t2[0:3,]

#View the t2_3 dataframe
t2_3
```

3 Section: Hands-On Analysis

22. Import the "adult_ch3_training" data set using the "Heading: Yes" setting. Rename the data set "adult" once it is imported.

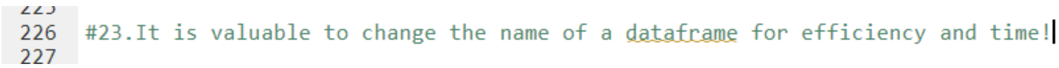


```

#Import adult_ch3.training.csv and name as adult
adult <- read.csv("adult_ch3_training.csv", header = TRUE)

```



```
In [54]:  # Manually upload the 'adult_ch3_training.csv' into folder
          # then call csv file and store as adult
          adult = pd.read_csv('adult_ch3_training.csv')
          # View the first five rows of data from df
          adult.head(5)

Out[54]:
             age  workclass  education        marital-  occupation     sex  capital-  capital-
                                              status                          gain      loss
```

23. Write a comment explaining the change in the data set name.

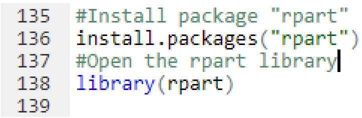


```
#23.It is valuable to change the name of a dataframe for efficiency and time!
```

24. Import the following packages:

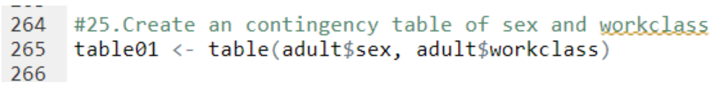
(1) For Python, import the *DecisionTreeClassifier* command from the tree package.

(2) For R, import the *rpart*. Make sure you both install and open the package.

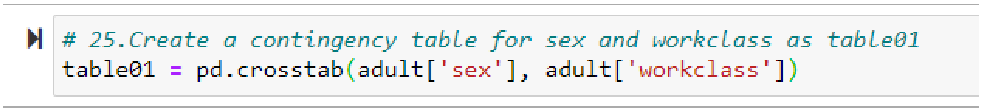


```
#Install package "rpart"
install.packages("rpart")
#Open the rpart library
library(rpart)
```

25. Create a contingency table of *sex* and *workclass* and save the output as ”table01”.

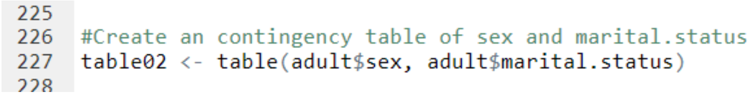


```
#25.Create an contingency table of sex and workclass
table01 <- table(adult$sex, adult$workclass)
```

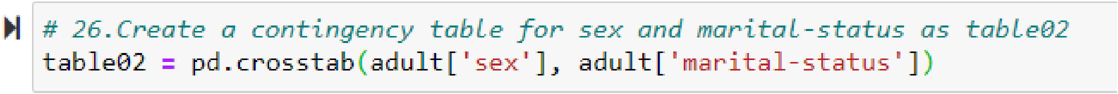


```
# 25.Create a contingency table for sex and workclass as table01
table01 = pd.crosstab(adult['sex'], adult['workclass'])
```

26. Create a contingency table of *sex* and *marital.status*. Save the output as ”table02”.

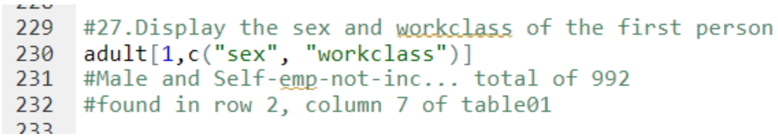


```
#Create an contingency table of sex and marital.status
table02 <- table(adult$sex, adult$marital.status)
```

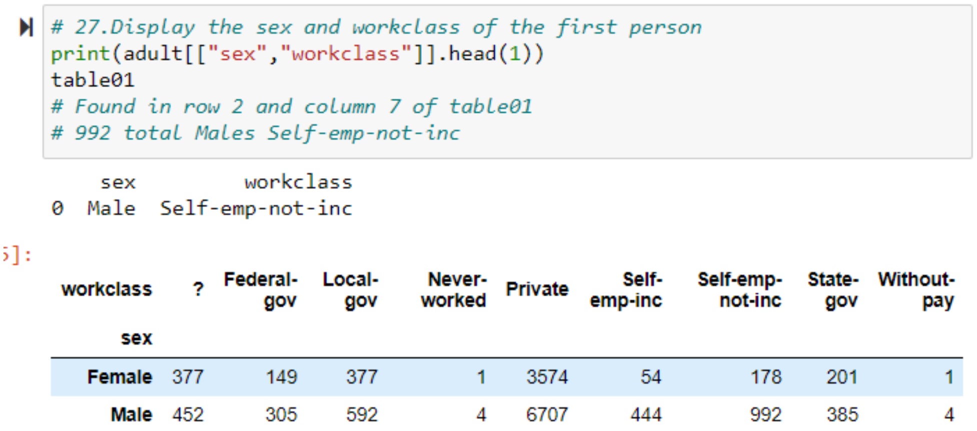


```
# 26.Create a contingency table for sex and marital-status as table02
table02 = pd.crosstab(adult['sex'], adult['marital-status'])
```

27. Display the *sex* and *workclass* values of the person in the first record. What cell of ”table01” do they belong to? How many other records in the data set have the same *sex* and *workclass* values?



```
#27.Display the sex and workclass of the first person
adult[1,c("sex", "workclass")]
#Male and Self-emp-not-inc... total of 992
#found in row 2, column 7 of table01
```

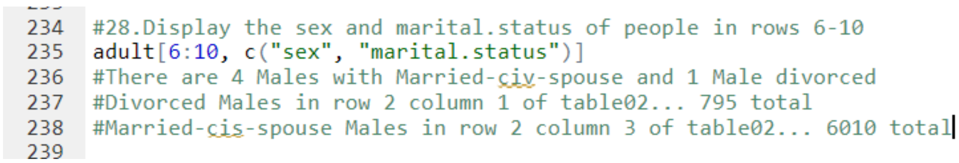


```python
# 27.Display the sex and workclass of the first person
print(adult[["sex","workclass"]].head(1))
table01
# Found in row 2 and column 7 of table01
# 992 total Males Self-emp-not-inc
```

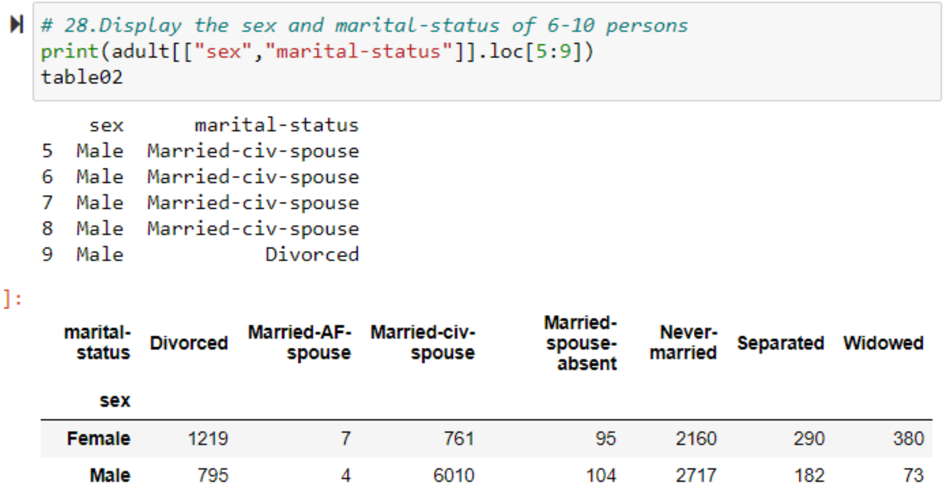
```
    sex         workclass
0  Male  Self-emp-not-inc
```

workclass	?	Federal-gov	Local-gov	Never-worked	Private	Self-emp-inc	Self-emp-not-inc	State-gov	Without-pay
sex									
Female	377	149	377	1	3574	54	178	201	1
Male	452	305	592	4	6707	444	992	385	4

28. Display the *sex* and *marital.status* values of the people in records 6-10. Which cells of ”table02” do they belong to? How many other records in the data set have the same combinations of *sex* and *marital.status* values?



```r
#28.Display the sex and marital.status of people in rows 6-10
adult[6:10, c("sex", "marital.status")]
#There are 4 Males with Married-civ-spouse and 1 Male divorced
#Divorced Males in row 2 column 1 of table02... 795 total
#Married-cis-spouse Males in row 2 column 3 of table02... 6010 total
```

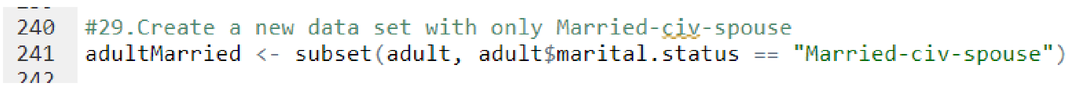


```python
# 28.Display the sex and marital-status of 6-10 persons
print(adult[["sex","marital-status"]].loc[5:9])
table02
```

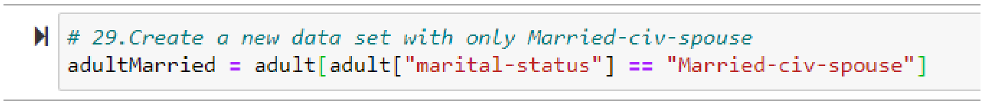
```
    sex      marital-status
5  Male  Married-civ-spouse
6  Male  Married-civ-spouse
7  Male  Married-civ-spouse
8  Male  Married-civ-spouse
9  Male            Divorced
```

marital-status	Divorced	Married-AF-spouse	Married-civ-spouse	Married-spouse-absent	Never-married	Separated	Widowed
sex							
Female	1219	7	761	95	2160	290	380
Male	795	4	6010	104	2717	182	73

29. Create a new data set that has only records whose *marital.status* is “Married-civ-spouse” and name the data set ”adultMarried”.

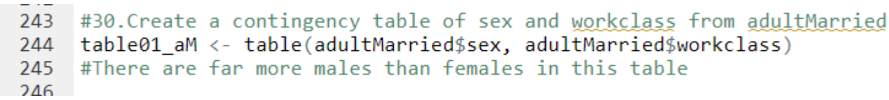


```r
#29.Create a new data set with only Married-civ-spouse
adultMarried <- subset(adult, adult$marital.status == "Married-civ-spouse")
```

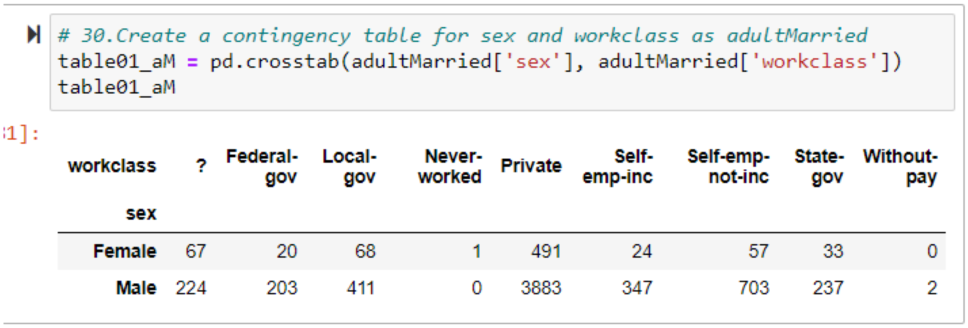


```python
# 29.Create a new data set with only Married-civ-spouse
adultMarried = adult[adult["marital-status"] == "Married-civ-spouse"]
```

30. Recreate the contingency table of *sex* and *workclass* using the "adultMarried" data set. What differences do you notice between the sexes?



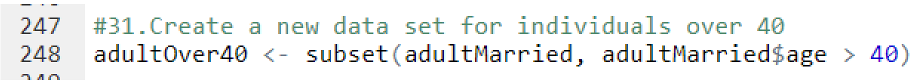
```
#30.Create a contingency table of sex and workclass from adultMarried
table01_aM <- table(adultMarried$sex, adultMarried$workclass)
#There are far more males than females in this table
```



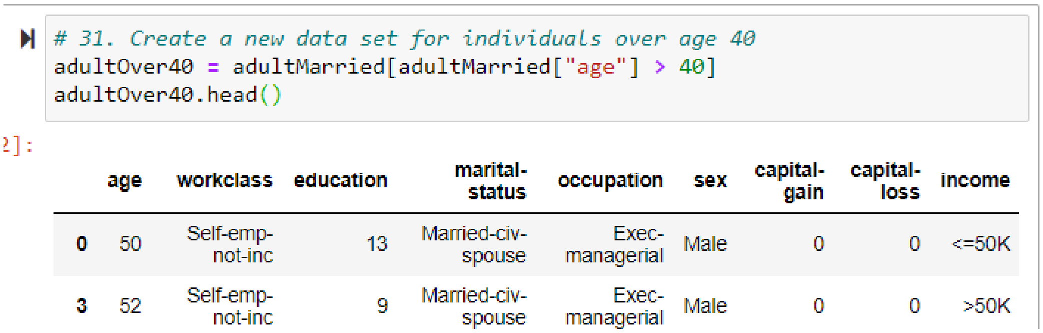
```
# 30.Create a contingency table for sex and workclass as adultMarried
table01_aM = pd.crosstab(adultMarried['sex'], adultMarried['workclass'])
table01_aM
```

workclass	?	Federal-gov	Local-gov	Never-worked	Private	Self-emp-inc	Self-emp-not-inc	State-gov	Without-pay
sex									
Female	67	20	68	1	491	24	57	33	0
Male	224	203	411	0	3883	347	703	237	2

31. Create a new data set that has only records whose age value is greater than 40. Name the new data set "adultOver40".



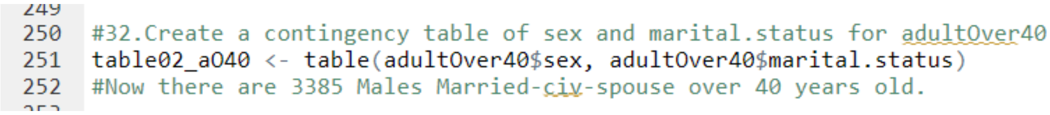
```
#31.Create a new data set for individuals over 40
adultOver40 <- subset(adultMarried, adultMarried$age > 40)
```



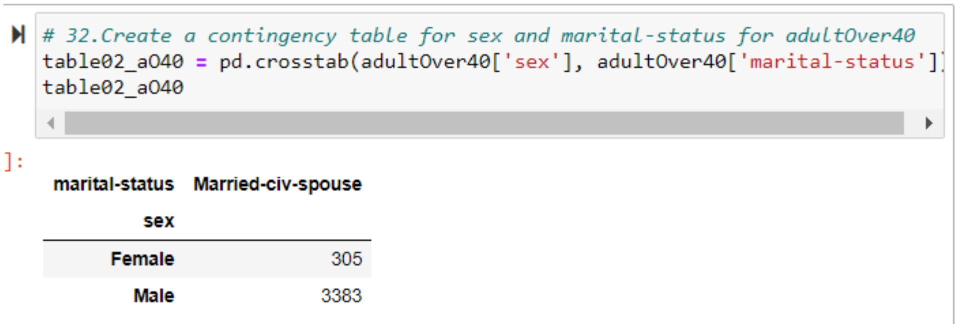
```
# 31. Create a new data set for individuals over age 40
adultOver40 = adultMarried[adultMarried["age"] > 40]
adultOver40.head()
```

	age	workclass	education	marital-status	occupation	sex	capital-gain	capital-loss	income
0	50	Self-emp-not-inc	13	Married-civ-spouse	Exec-managerial	Male	0	0	<=50K
3	52	Self-emp-not-inc	9	Married-civ-spouse	Exec-managerial	Male	0	0	>50K

32. Recreate a contingency table of *sex* and *marital status* using the "adultOver40" data set. What differences do you notice?



```
#32.Create a contingency table of sex and marital.status for adultOver40
table02_aO40 <- table(adultOver40$sex, adultOver40$marital.status)
#Now there are 3385 Males Married-civ-spouse over 40 years old.
```



```
# 32.Create a contingency table for sex and marital-status for adultOver40
table02_aO40 = pd.crosstab(adultOver40['sex'], adultOver40['marital-status']
table02_aO40
```

marital-status	Married-civ-spouse
sex	
Female	305
Male	3383

4 Section: Concepts

33. Describe two reasons why it might be a good idea to add an index field to the data set.
An index field is valuable in a data set because it assigns a number to a row or column making it easier to call specific parts of the data. This allows for subsets to be created as well using an index field.

34. Explain why the field *days_since_previous* is essentially useless until we handle the 999 code.
The 999 code is a way to communicate that there is no value or data collected. These values are outliers in the data set. It is essential to convert these outliers useless values to ”NA” in R or ”NaN” in Python for them to not impact the data and visualizations.

35. Why was it important to re-express *education* as a numeric field?
Numeric fields can be calculated on, whereas, character fields cannot have functions coded with them. Changing *education* to numeric allows us to use functions within that field.

36. Suppose a data value has a *z-value* of 1. How may we interpret this value?
z-values are used to help identify outliers with a scaled value. A 1 would be interpreted as a value that is not an outlier.

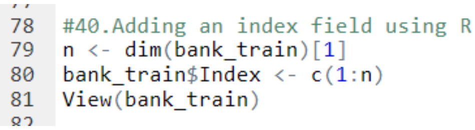
37. What is the rough rule of thumb for identifying outliers using z-values?
If z-values are < -3 or > 3 then they are considered outliers.

38. Should outliers be automatically removed or changed? Why or why not?
I personally do not think that outliers should be removed! I do think that, depending on the analytics that need to be made, outliers can be changed. However, outliers have a reason and a purpose and it is important to know what they are and why and then note it in the explanation of the data.

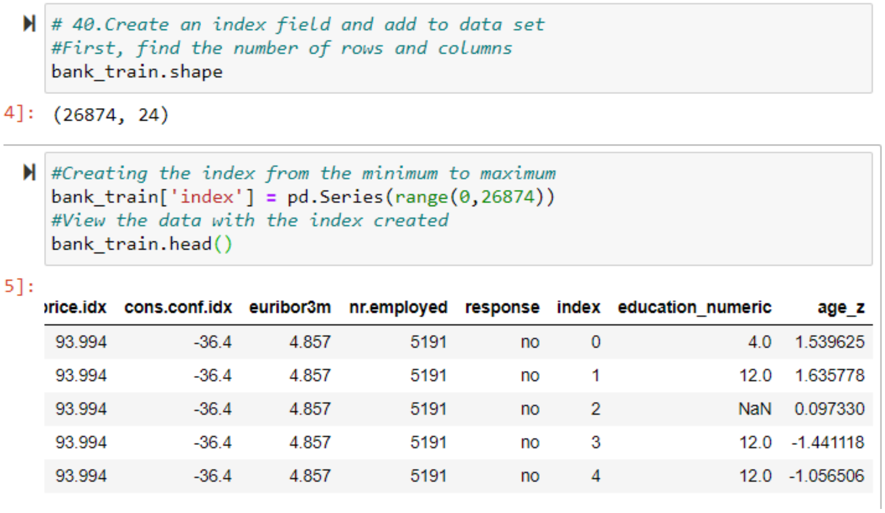
39. What should we do with outliers we have identified?
We can change the value to ”NA” in R or ”NaN” in Python like explained, or they can be ignored through code, or you can delete them... Not the best idea.

5 Section: Working with the Data

40. Derive an index field and add it to the data set.



```
#40.Adding an index field using R
n <- dim(bank_train)[1]
bank_train$Index <- c(1:n)
View(bank_train)
```



```
# 40.Create an index field and add to data set
#First, find the number of rows and columns
bank_train.shape
```

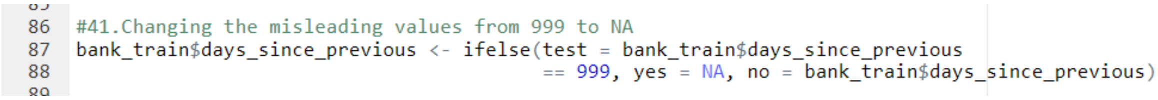
```
4]: (26874, 24)
```

```
#Creating the index from the minimum to maximum
bank_train['index'] = pd.Series(range(0,26874))
#View the data with the index created
bank_train.head()
```

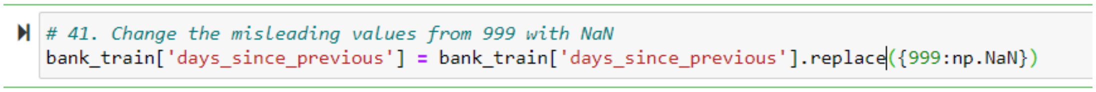
5]:

rice.idx	cons.conf.idx	euribor3m	nr.employed	response	index	education_numeric	age_z
93.994	-36.4	4.857	5191	no	0	4.0	1.539625
93.994	-36.4	4.857	5191	no	1	12.0	1.635778
93.994	-36.4	4.857	5191	no	2	NaN	0.097330
93.994	-36.4	4.857	5191	no	3	12.0	-1.441118
93.994	-36.4	4.857	5191	no	4	12.0	-1.056506

41. For the *days_since_previous* field, change the field value 999 to the appropriate code for missing values, as shown in class.

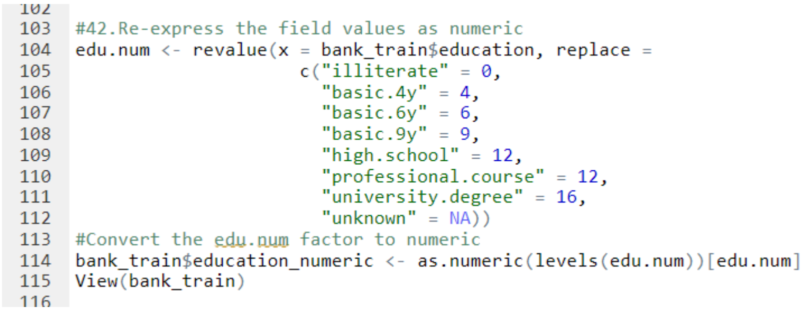


```
#41.Changing the misleading values from 999 to NA
bank_train$days_since_previous <- ifelse(test = bank_train$days_since_previous
                                        == 999, yes = NA, no = bank_train$days_since_previous)
```

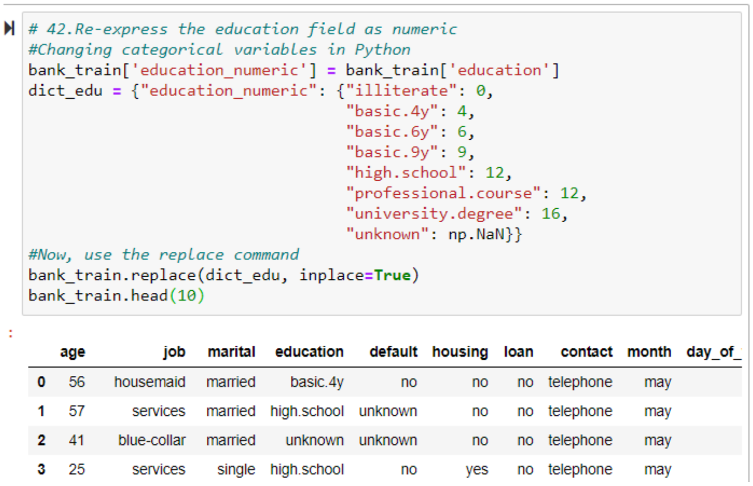


```
# 41. Change the misleading values from 999 with NaN
bank_train['days_since_previous'] = bank_train['days_since_previous'].replace({999:np.NaN})
```

42. For the *education* field, re-express the field values as the numeric values shown in class.



```
#42.Re-express the field values as numeric
edu.num <- revalue(x = bank_train$education, replace =
                     c("illiterate" = 0,
                       "basic.4y" = 4,
                       "basic.6y" = 6,
                       "basic.9y" = 9,
                       "high.school" = 12,
                       "professional.course" = 12,
                       "university.degree" = 16,
                       "unknown" = NA))
#Convert the edu.num factor to numeric
bank_train$education_numeric <- as.numeric(levels(edu.num))[edu.num]
View(bank_train)
```

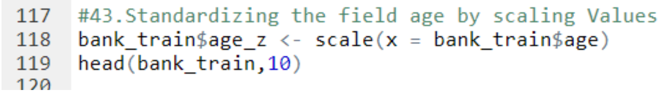


```
# 42.Re-express the education field as numeric
#Changing categorical variables in Python
bank_train['education_numeric'] = bank_train['education']
dict_edu = {"education_numeric": {"illiterate": 0,
                                  "basic.4y": 4,
                                  "basic.6y": 6,
                                  "basic.9y": 9,
                                  "high.school": 12,
                                  "professional.course": 12,
                                  "university.degree": 16,
                                  "unknown": np.NaN}}
#Now, use the replace command
bank_train.replace(dict_edu, inplace=True)
bank_train.head(10)
```

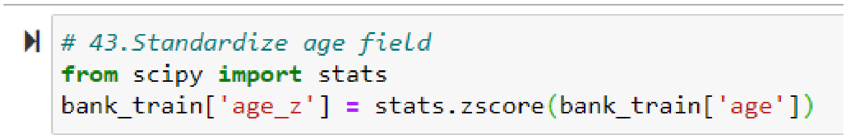
:

	age	job	marital	education	default	housing	loan	contact	month	day_of_
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	41	blue-collar	married	unknown	unknown	no	no	telephone	may	
3	25	services	single	high.school	no	yes	no	telephone	may	

43. Standardize the field *age*. Print out a list of the first 10 records, including the variables *age* and *age_z*.

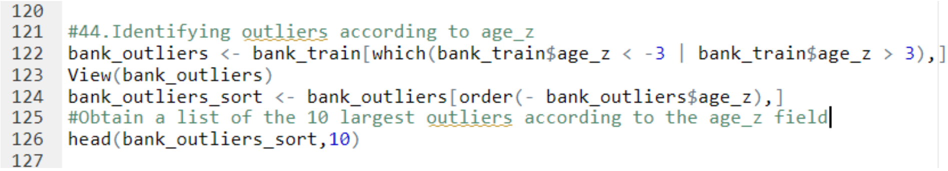


```
#43.Standardizing the field age by scaling Values
bank_train$age_z <- scale(x = bank_train$age)
head(bank_train,10)
```

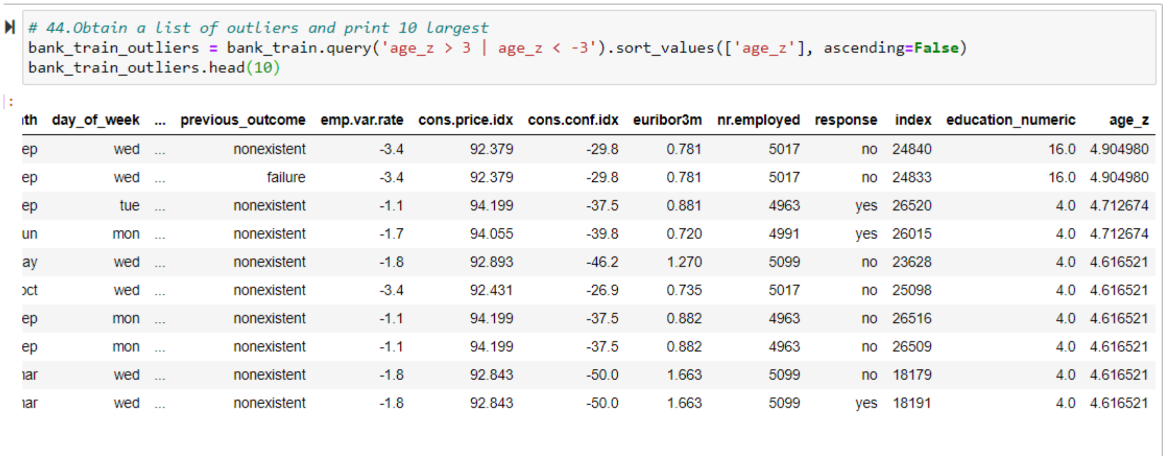


```
# 43.Standardize age field
from scipy import stats
bank_train['age_z'] = stats.zscore(bank_train['age'])
```

44. Obtain a listing of all records that are outliers according to the field *age_z*. Print out a listing of the 10 largest *age_z*.



```
#44.Identifying outliers according to age_z
bank_outliers <- bank_train[which(bank_train$age_z < -3 | bank_train$age_z > 3),]
View(bank_outliers)
bank_outliers_sort <- bank_outliers[order(- bank_outliers$age_z),]
#Obtain a list of the 10 largest outliers according to the age_z field
head(bank_outliers_sort,10)
```

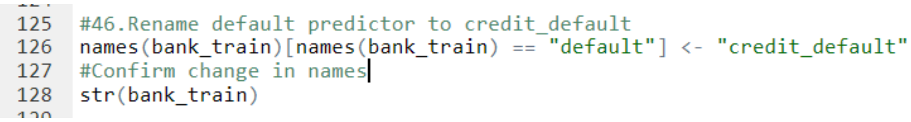


```
# 44.Obtain a list of outliers and print 10 largest
bank_train_outliers = bank_train.query('age_z > 3 | age_z < -3').sort_values(['age_z'], ascending=False)
bank_train_outliers.head(10)
```

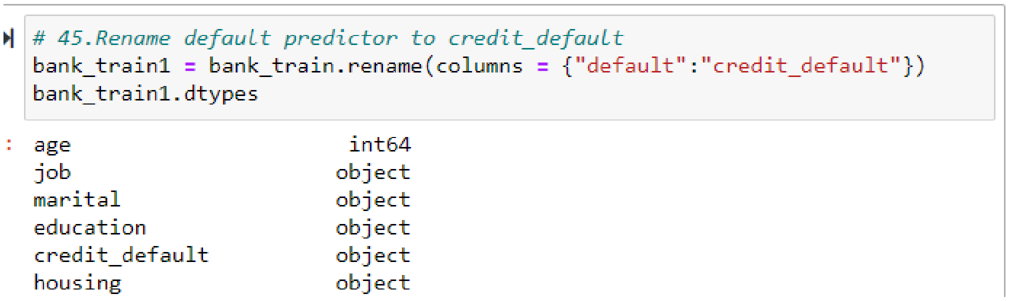
ıth	day_of_week	...	previous_outcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	response	index	education_numeric	age_z
ep	wed	...	nonexistent	-3.4	92.379	-29.8	0.781	5017	no	24840	16.0	4.904980
ep	wed	...	failure	-3.4	92.379	-29.8	0.781	5017	no	24833	16.0	4.904980
ep	tue	...	nonexistent	-1.1	94.199	-37.5	0.881	4963	yes	26520	4.0	4.712674
un	mon	...	nonexistent	-1.7	94.055	-39.8	0.720	4991	yes	26015	4.0	4.712674
ay	wed	...	nonexistent	-1.8	92.893	-46.2	1.270	5099	no	23628	4.0	4.616521
ɔct	wed	...	nonexistent	-3.4	92.431	-26.9	0.735	5017	no	25098	4.0	4.616521
ep	mon	...	nonexistent	-1.1	94.199	-37.5	0.882	4963	no	26516	4.0	4.616521
ep	mon	...	nonexistent	-1.1	94.199	-37.5	0.882	4963	no	26509	4.0	4.616521
ıar	wed	...	nonexistent	-1.8	92.843	-50.0	1.663	5099	no	18179	4.0	4.616521
ıar	wed	...	nonexistent	-1.8	92.843	-50.0	1.663	5099	yes	18191	4.0	4.616521

45. For the *job* field, combine the jobs with less than 5% of the records into a field called *other*.

46. Rename the *default* predictor to *credit_default*.



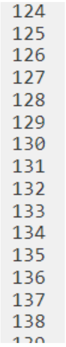
```
#46.Rename default predictor to credit_default
names(bank_train)[names(bank_train) == "default"] <- "credit_default"
#Confirm change in names
str(bank_train)
```



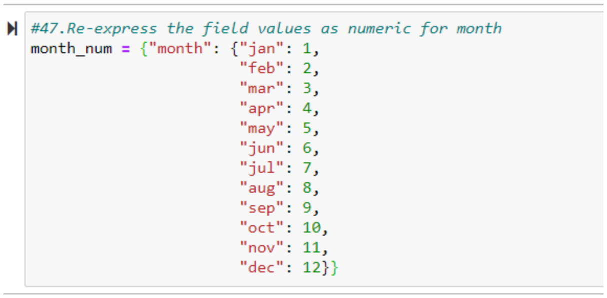
```
# 45.Rename default predictor to credit_default
bank_train1 = bank_train.rename(columns = {"default":"credit_default"})
bank_train1.dtypes
```

```
age                int64
job               object
marital           object
education         object
credit_default    object
housing           object
```

47. For the variable *month*, change the field values to $1 - 12$, but keep the variable as categorical.



```

#47.Re-express the field values as numeric for month
month_num <- revalue(x = bank_train$month, replace =
                       c("jan" = 1,
                         "feb" = 2,
                         "mar" = 3,
                         "apr" = 4,
                         "may" = 5,
                         "jun" = 6,
                         "jul" = 7,
                         "aug" = 8,
                         "sep" = 9,
                         "oct" = 10,
                         "nov" = 11,
                         "dec" = 12))
```

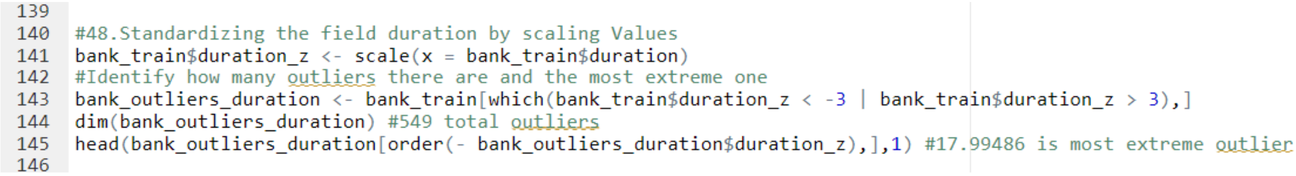


```
#47.Re-express the field values as numeric for month
month_num = {"month": {"jan": 1,
                       "feb": 2,
                       "mar": 3,
                       "apr": 4,
                       "may": 5,
                       "jun": 6,
                       "jul": 7,
                       "aug": 8,
                       "sep": 9,
                       "oct": 10,
                       "nov": 11,
                       "dec": 12}}
```

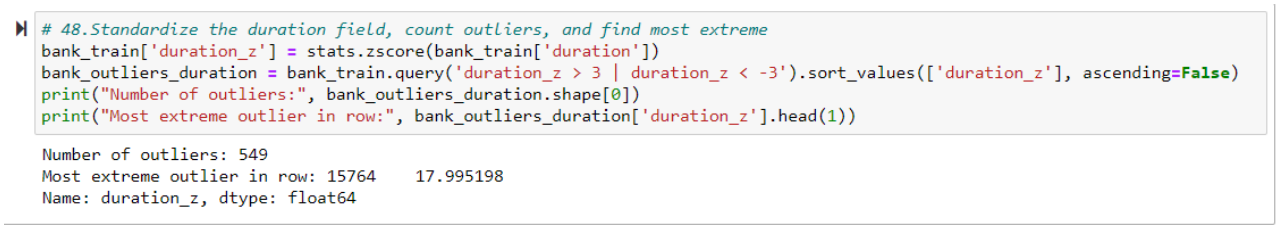
48. Do the following for the *duration*

(1) Standardize the variable.

(2) Identify how many outliers there are and identify the most extreme outlier.



```

#48.Standardizing the field duration by scaling Values
bank_train$duration_z <- scale(x = bank_train$duration)
#Identify how many outliers there are and the most extreme one
bank_outliers_duration <- bank_train[which(bank_train$duration_z < -3 | bank_train$duration_z > 3),]
dim(bank_outliers_duration) #549 total outliers
head(bank_outliers_duration[order(- bank_outliers_duration$duration_z),],1) #17.99486 is most extreme outlier

```



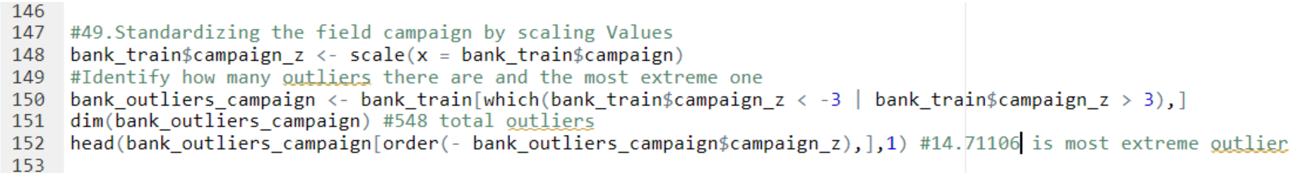
```
# 48.Standardize the duration field, count outliers, and find most extreme
bank_train['duration_z'] = stats.zscore(bank_train['duration'])
bank_outliers_duration = bank_train.query('duration_z > 3 | duration_z < -3').sort_values(['duration_z'], ascending=False)
print("Number of outliers:", bank_outliers_duration.shape[0])
print("Most extreme outlier in row:", bank_outliers_duration['duration_z'].head(1))
```

```
Number of outliers: 549
Most extreme outlier in row: 15764    17.995198
Name: duration_z, dtype: float64
```

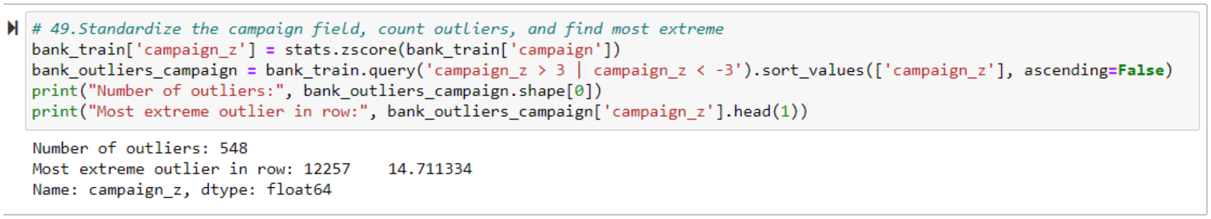
49. Do the following for the *campaign*

(1) Standardize the variable.

(2) Identify how many outliers there are and identify the most extreme outlier.



```

#49.Standardizing the field campaign by scaling Values
bank_train$campaign_z <- scale(x = bank_train$campaign)
#Identify how many outliers there are and the most extreme one
bank_outliers_campaign <- bank_train[which(bank_train$campaign_z < -3 | bank_train$campaign_z > 3),]
dim(bank_outliers_campaign) #548 total outliers
head(bank_outliers_campaign[order(- bank_outliers_campaign$campaign_z),],1) #14.71106 is most extreme outlier

```



```
# 49.Standardize the campaign field, count outliers, and find most extreme
bank_train['campaign_z'] = stats.zscore(bank_train['campaign'])
bank_outliers_campaign = bank_train.query('campaign_z > 3 | campaign_z < -3').sort_values(['campaign_z'], ascending=False)
print("Number of outliers:", bank_outliers_campaign.shape[0])
print("Most extreme outlier in row:", bank_outliers_campaign['campaign_z'].head(1))
```

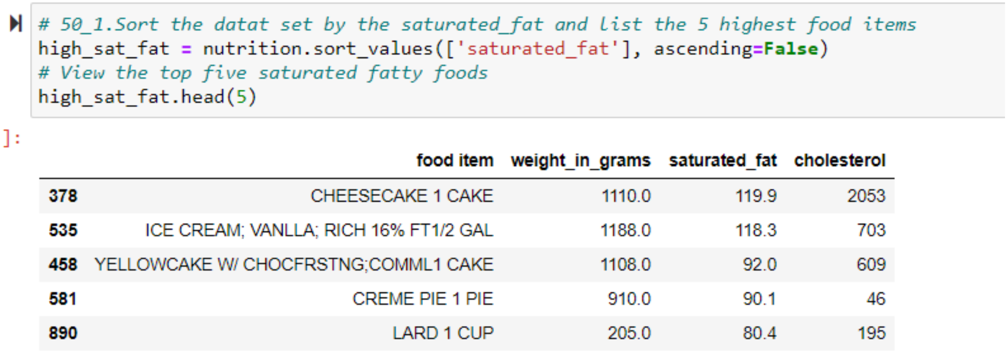
```
Number of outliers: 548
Most extreme outlier in row: 12257    14.711334
Name: campaign_z, dtype: float64
```

6 Section: Hands-On Analysis

For questions 50 − 53, work with the "Nutrition_subset" data set. The data set contains the weight in grams along with the amount of saturated fat and the amount of cholesterol for a set of 961 foods. Use either Python or R to solve each problem.

50. The elements in the data set are food items of various sizes, ranging from a teaspoon of cinnamon to an entire carrot cake.

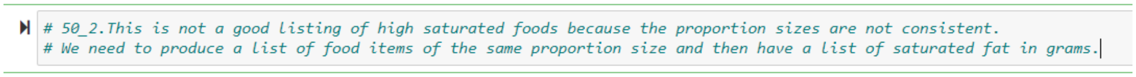
    (1) Sort the data set by the saturated fat (*saturated_fat*) and produce a listing of the five food items highest in saturated fat.



```
# 50_1.Sort the datat set by the saturated_fat and list the 5 highest food items
high_sat_fat = nutrition.sort_values(['saturated_fat'], ascending=False)
# View the top five saturated fatty foods
high_sat_fat.head(5)
```

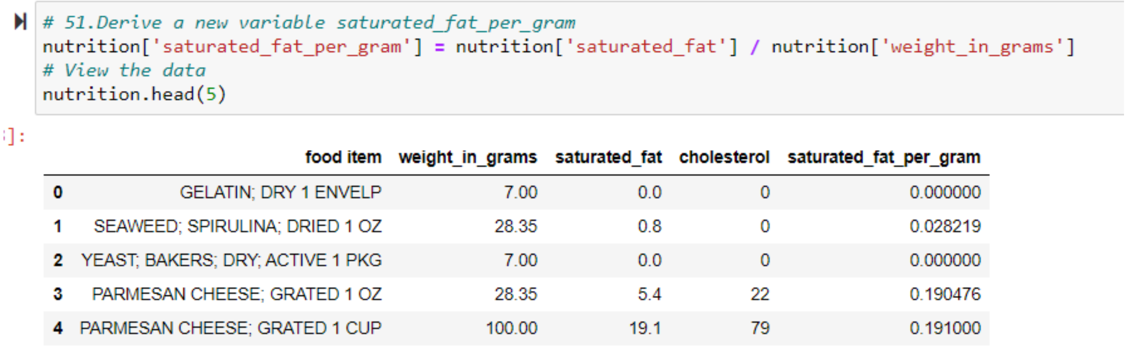
	food item	weight_in_grams	saturated_fat	cholesterol
378	CHEESECAKE 1 CAKE	1110.0	119.9	2053
535	ICE CREAM; VANLLA; RICH 16% FT1/2 GAL	1188.0	118.3	703
458	YELLOWCAKE W/ CHOCFRSTNG;COMML1 CAKE	1108.0	92.0	609
581	CREME PIE 1 PIE	910.0	90.1	46
890	LARD 1 CUP	205.0	80.4	195

    (2) Comment on the validity of comparing food items of different sizes.



```
# 50_2.This is not a good listing of high saturated foods because the proportion sizes are not consistent.
# We need to produce a list of food items of the same proportion size and then have a list of saturated fat in grams.
```

51. Derive a new variable, *saturated_fat_per_gram*, by dividing the amount of saturated fat by the weight in grams.

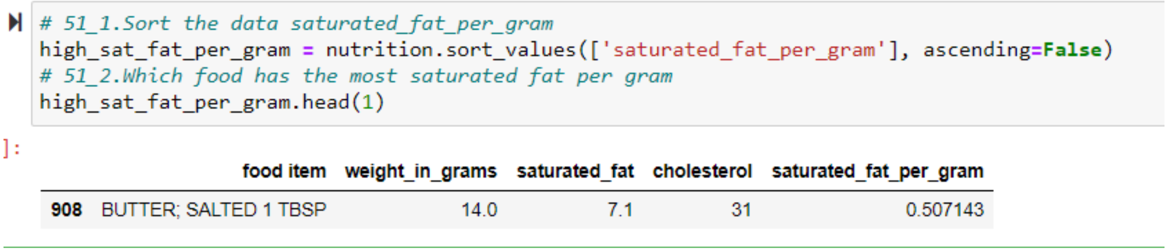


```
# 51.Derive a new variable saturated_fat_per_gram
nutrition['saturated_fat_per_gram'] = nutrition['saturated_fat'] / nutrition['weight_in_grams']
# View the data
nutrition.head(5)
```

	food item	weight_in_grams	saturated_fat	cholesterol	saturated_fat_per_gram
0	GELATIN; DRY 1 ENVELP	7.00	0.0	0	0.000000
1	SEAWEED; SPIRULINA; DRIED 1 OZ	28.35	0.8	0	0.028219
2	YEAST; BAKERS; DRY; ACTIVE 1 PKG	7.00	0.0	0	0.000000
3	PARMESAN CHEESE; GRATED 1 OZ	28.35	5.4	22	0.190476
4	PARMESAN CHEESE; GRATED 1 CUP	100.00	19.1	79	0.191000

(1) Sort the data set by *saturated_fat_per_gram*, by dividing the amount of saturated fat by the weight in grams.

(2) Which food has the most saturated fat per gram?



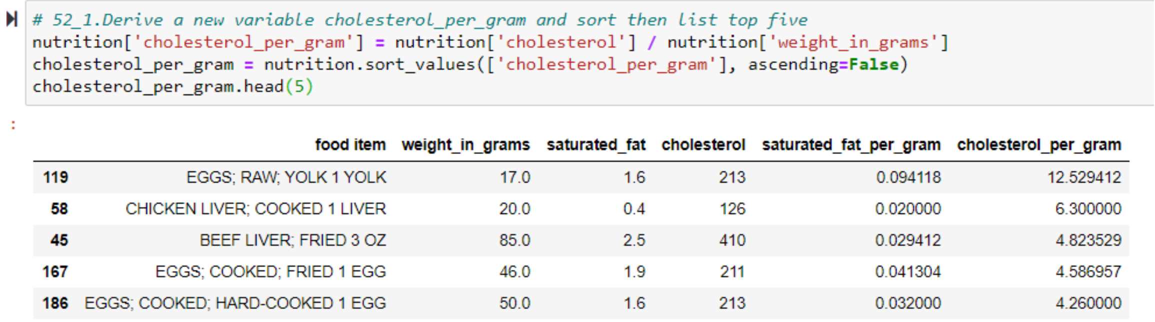
```
# 51_1.Sort the data saturated_fat_per_gram
high_sat_fat_per_gram = nutrition.sort_values(['saturated_fat_per_gram'], ascending=False)
# 51_2.Which food has the most saturated fat per gram
high_sat_fat_per_gram.head(1)
```

]:

	food item	weight_in_grams	saturated_fat	cholesterol	saturated_fat_per_gram
908	BUTTER; SALTED 1 TBSP	14.0	7.1	31	0.507143

52. Derive a new variable,

(1) Sort the data set by *cholesterol_per_gram* and produce a listing of the five food items highest in cholesterol fat per gram.

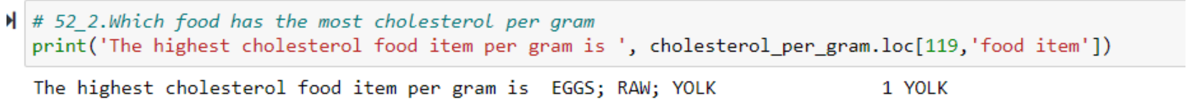


```
# 52_1.Derive a new variable cholesterol_per_gram and sort then list top five
nutrition['cholesterol_per_gram'] = nutrition['cholesterol'] / nutrition['weight_in_grams']
cholesterol_per_gram = nutrition.sort_values(['cholesterol_per_gram'], ascending=False)
cholesterol_per_gram.head(5)
```

:

	food item	weight_in_grams	saturated_fat	cholesterol	saturated_fat_per_gram	cholesterol_per_gram
119	EGGS; RAW; YOLK 1 YOLK	17.0	1.6	213	0.094118	12.529412
58	CHICKEN LIVER; COOKED 1 LIVER	20.0	0.4	126	0.020000	6.300000
45	BEEF LIVER; FRIED 3 OZ	85.0	2.5	410	0.029412	4.823529
167	EGGS; COOKED; FRIED 1 EGG	46.0	1.9	211	0.041304	4.586957
186	EGGS; COOKED; HARD-COOKED 1 EGG	50.0	1.6	213	0.032000	4.260000

(2) Which food has the most cholesterol fat per gram?

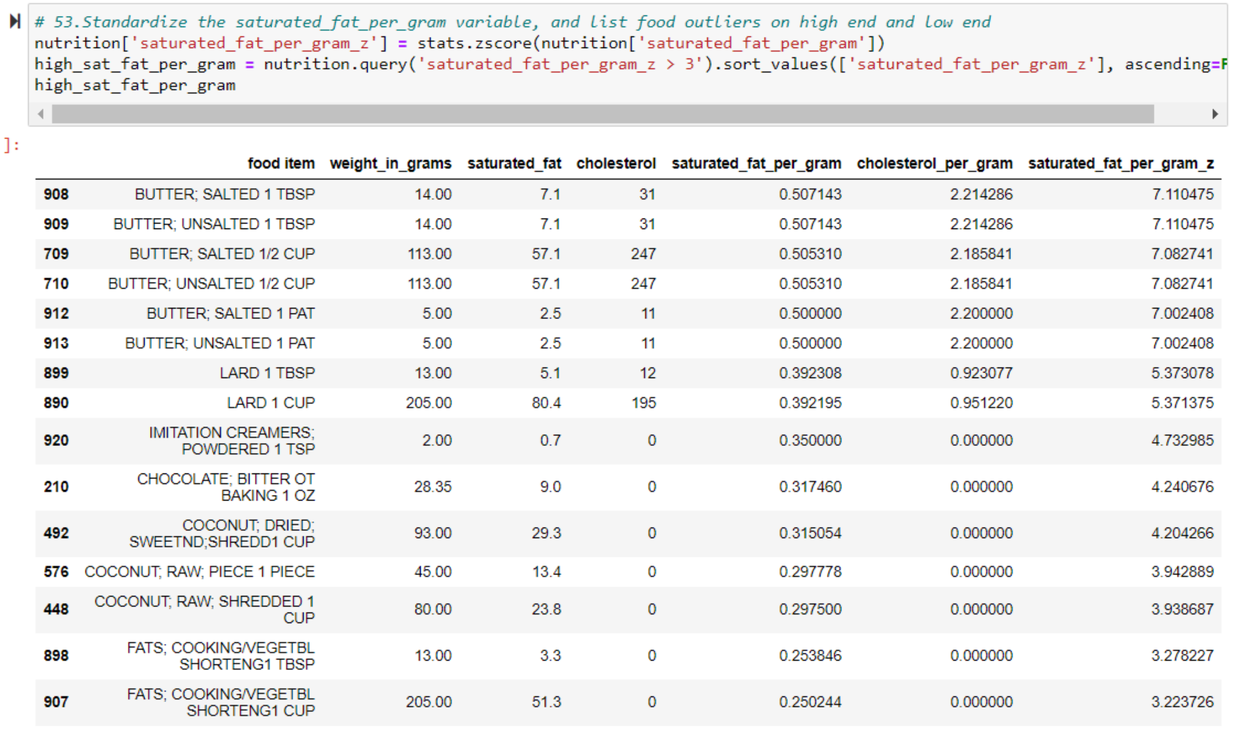


```
# 52_2.Which food has the most cholesterol per gram
print('The highest cholesterol food item per gram is ', cholesterol_per_gram.loc[119,'food item'])
```

```
The highest cholesterol food item per gram is  EGGS; RAW; YOLK                1 YOLK
```

53. Standardize the field *saturated_fat_per_gram*. Produce a listing of all the food items that are outliers at the high end of the scale. How many food items are outliers at the low end of the

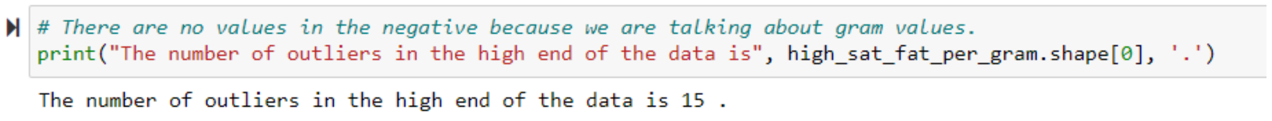
scale?



```
# 53.Standardize the saturated_fat_per_gram variable, and list food outliers on high end and low end
nutrition['saturated_fat_per_gram_z'] = stats.zscore(nutrition['saturated_fat_per_gram'])
high_sat_fat_per_gram = nutrition.query('saturated_fat_per_gram_z > 3').sort_values(['saturated_fat_per_gram_z'], ascending=F
high_sat_fat_per_gram
```

]:

	food item	weight_in_grams	saturated_fat	cholesterol	saturated_fat_per_gram	cholesterol_per_gram	saturated_fat_per_gram_z
908	BUTTER; SALTED 1 TBSP	14.00	7.1	31	0.507143	2.214286	7.110475
909	BUTTER; UNSALTED 1 TBSP	14.00	7.1	31	0.507143	2.214286	7.110475
709	BUTTER; SALTED 1/2 CUP	113.00	57.1	247	0.505310	2.185841	7.082741
710	BUTTER; UNSALTED 1/2 CUP	113.00	57.1	247	0.505310	2.185841	7.082741
912	BUTTER; SALTED 1 PAT	5.00	2.5	11	0.500000	2.200000	7.002408
913	BUTTER; UNSALTED 1 PAT	5.00	2.5	11	0.500000	2.200000	7.002408
899	LARD 1 TBSP	13.00	5.1	12	0.392308	0.923077	5.373078
890	LARD 1 CUP	205.00	80.4	195	0.392195	0.951220	5.371375
920	IMITATION CREAMERS; POWDERED 1 TSP	2.00	0.7	0	0.350000	0.000000	4.732985
210	CHOCOLATE; BITTER OT BAKING 1 OZ	28.35	9.0	0	0.317460	0.000000	4.240676
492	COCONUT; DRIED; SWEETND;SHREDD1 CUP	93.00	29.3	0	0.315054	0.000000	4.204266
576	COCONUT; RAW; PIECE 1 PIECE	45.00	13.4	0	0.297778	0.000000	3.942889
448	COCONUT; RAW; SHREDDED 1 CUP	80.00	23.8	0	0.297500	0.000000	3.938687
898	FATS; COOKING/VEGETBL SHORTENG1 TBSP	13.00	3.3	0	0.253846	0.000000	3.278227
907	FATS; COOKING/VEGETBL SHORTENG1 CUP	205.00	51.3	0	0.250244	0.000000	3.223726

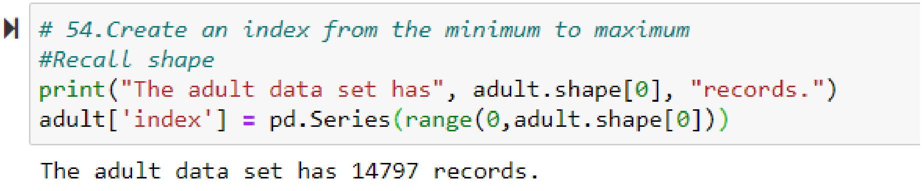


```
# There are no values in the negative because we are talking about gram values.
print("The number of outliers in the high end of the data is", high_sat_fat_per_gram.shape[0], '.')
```

```
The number of outliers in the high end of the data is 15 .
```

For questions 54 – 58, work with the *adult_ch3_training* data set. The response is whether income exceeds $50,000.

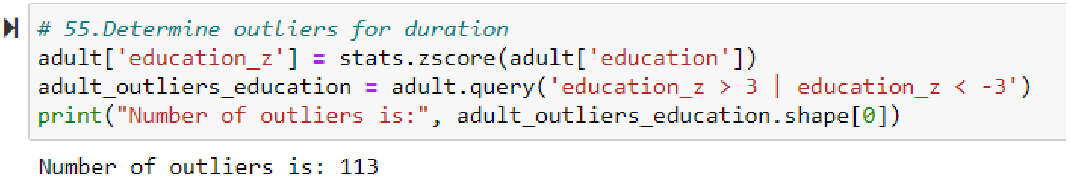
54. Add a record index field to the data set.



```
# 54.Create an index from the minimum to maximum
#Recall shape
print("The adult data set has", adult.shape[0], "records.")
adult['index'] = pd.Series(range(0,adult.shape[0]))
```

```
The adult data set has 14797 records.
```

55. Determine whether any outliers exist for the *education*.



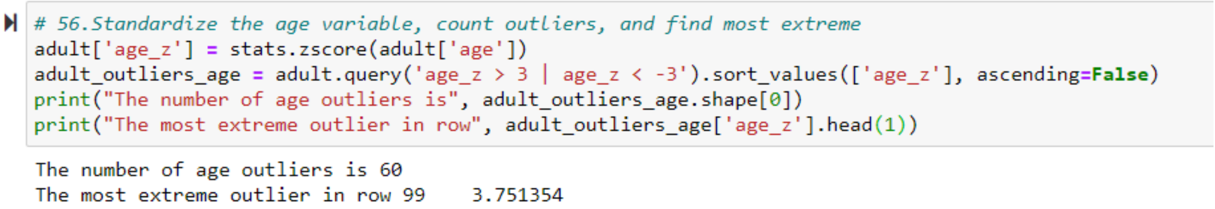
```
# 55.Determine outliers for duration
adult['education_z'] = stats.zscore(adult['education'])
adult_outliers_education = adult.query('education_z > 3 | education_z < -3')
print("Number of outliers is:", adult_outliers_education.shape[0])
```

```
Number of outliers is: 113
```

56. Do the following for the *age*.

(1) Standardize the variable.

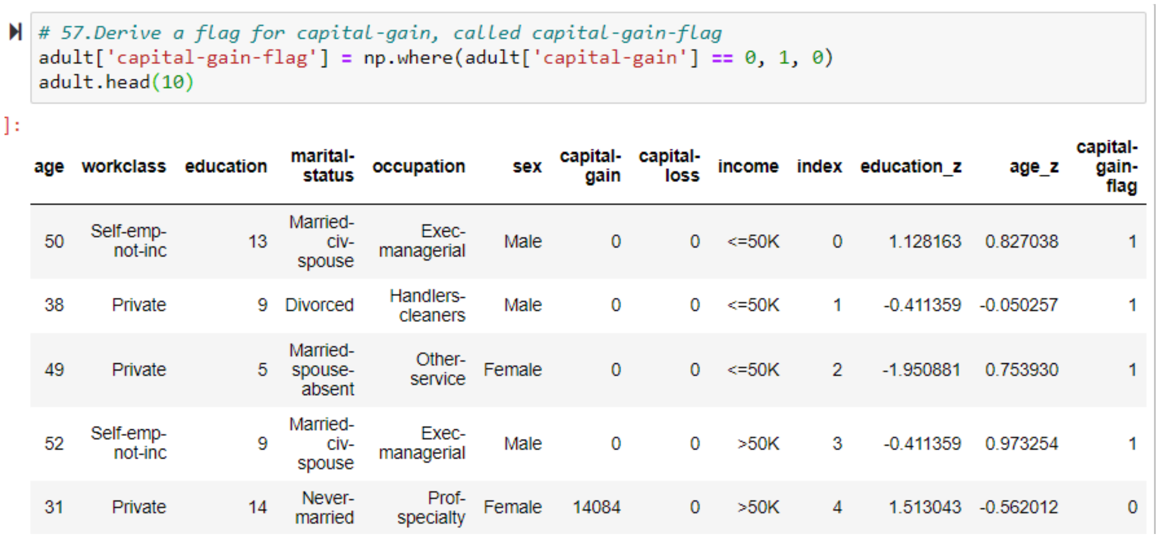
(2) Identify how many outliers there are and identify the most extreme outlier.



```
# 56.Standardize the age variable, count outliers, and find most extreme
adult['age_z'] = stats.zscore(adult['age'])
adult_outliers_age = adult.query('age_z > 3 | age_z < -3').sort_values(['age_z'], ascending=False)
print("The number of age outliers is", adult_outliers_age.shape[0])
print("The most extreme outlier in row", adult_outliers_age['age_z'].head(1))
```

```
The number of age outliers is 60
The most extreme outlier in row 99    3.751354
```

57. Derive a flag for *capital-gain*, called *capital-gain-flag*, which equals 0 for capital gain, and 1 otherwise.



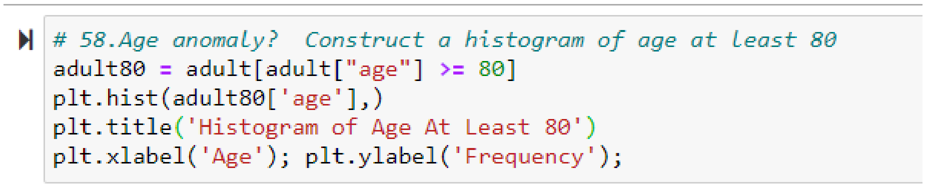
```
# 57.Derive a flag for capital-gain, called capital-gain-flag
adult['capital-gain-flag'] = np.where(adult['capital-gain'] == 0, 1, 0)
adult.head(10)
```

]:

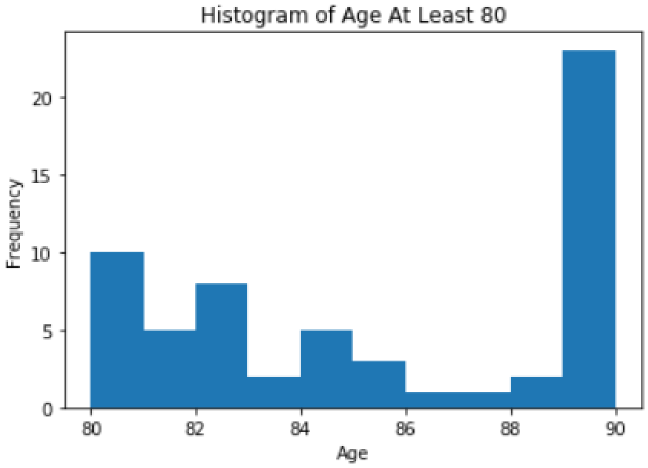
age	workclass	education	marital-status	occupation	sex	capital-gain	capital-loss	income	index	education_z	age_z	capital-gain-flag
50	Self-emp-not-inc	13	Married-civ-spouse	Exec-managerial	Male	0	0	<=50K	0	1.128163	0.827038	1
38	Private	9	Divorced	Handlers-cleaners	Male	0	0	<=50K	1	-0.411359	-0.050257	1
49	Private	5	Married-spouse-absent	Other-service	Female	0	0	<=50K	2	-1.950881	0.753930	1
52	Self-emp-not-inc	9	Married-civ-spouse	Exec-managerial	Male	0	0	>50K	3	-0.411359	0.973254	1
31	Private	14	Never-married	Prof-specialty	Female	14084	0	>50K	4	1.513043	-0.562012	0

58. Age anomaly? Select only records with age at least 80. Construct a histogram of age. Explain

what you see in one sentence and why it is like that in another sentence.



```
# 58.Age anomaly?  Construct a histogram of age at least 80
adult80 = adult[adult["age"] >= 80]
plt.hist(adult80['age'],)
plt.title('Histogram of Age At Least 80')
plt.xlabel('Age'); plt.ylabel('Frequency');
```



We have peak of records with age 89 compared to frequencies half the size or less from 80-88, with no records older than 90. This may be that anyone 89 and older are grouped together into one age category.

7 Section: Tableau

59. Create an interactive Tableau dashboard for the SLC restaurants with an embedded Google map.
Visit Magon's Tableau Public for this portion.