Data and Visual Analytics Lab

Lab Manual with Student Lab Record

Developed By

Dr. K. Rajkumar

Dr. B. Karthikeyan

Roll No	
Name	
Class	**I MSc Data Science**

Department of Data Science
Bishop Heber College (Autonomous)
Tiruchirappalli 620017, INDIA

March 2021

Department of Data Science
Bishop Heber College (Autonomous)
Tiruchirappalli 620017

BONAFIDE CERTIFICATE

Name:______________________________________

Reg. No:__________________________________ **Class:**________________

Course Title ____

Certified that this is the bonafide record of work done by me during **Odd / Even** Semester of **2020 – 2021** and submitted for the Practical Examination on ______________________________

Staff In-Charge **Head of the Department**

Examiners

1._______________________________________

2._______________________________________

Grade Sheet

Roll No		Name	
Year		Semester	
Instructor Name			

Lab	Date	Activity	Grade
1		Wine Data Analytics using NumPy Part-I	
2		Wine Data Analytics using NumPy Part-II	
3		Pandas Indexing and Selection	
4		Pandas Grouping and Aggregation	
5		Pandas Concatenation, Merging and Join	
6		Data Cleaning in Pandas	
7		Data Visualization using Seaborn	
8		Pandas Time Series Analysis	
9		Exploratory Data Analysis on Cardiovascular Data	
10		Advanced Data Wrangling in Pandas	
11		Interactive Dashboard Creation in Tableau	

Preface

This laboratory manual is written to accompany the lab course titled, *Data and Visual Analytics Lab*. The aim of this laboratory manual is to help students to enhance the understanding of concepts presented in class and to solve problems outlined in the syllabus of a lab course.

The lab exercises have been grouped into weekly activities for a semester. The weekly lab sheets include: input, output, source code and extra credit activities.

Students using these lab sheets should note the following:

1. Fill out your roll no and name in all required places.
2. Read carefully all details of an exercise of a week.
3. Understand the source code which may be a complete code or just a code snippet.
4. The required updates would have been included as comments inside the source code. You need to update them so that your code is ready for execution.
5. Once your code is executable, run your code with the test case inputs and get the results. Verify your obtained output against the expected output.
6. Now, carefully read all extra credit activities, revise your code accordingly, rerun your source code and obtain new outputs.
7. Upon solving all exercises including extra credit activities, approach your lab instructor, demonstrate your experiments and get your grades for the lab.

Final note, attend your weekly lab session with your lab manual without fail. Also, it is your responsibility to keep your lab manual safe as it records the grade you received every week. Comments about the laboratory exercises presented in this Lab manual are welcomed and encouraged. We hope that you will overlook any misspellings, omissions, errors and inconsistencies and report such issues to us. Happy coding!

Dr. K. Rajkumar

Department of Data Science - Data and Visual Analytics Lab

Lab1.Red Wine Quality Data Analytics using NumPy Part-I

Objectives

In this lab, you will learn the basics of NumPy.

How to Use This Jupyter Notebook

For each question, you should write NumPy statements in the "In[]" Cell and the expected output "Out[]" is already shown just below all In[] cells.

In [1]:
```
'''
Wine quality dataset 11 input features and 1 output feature

1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
Output variable (based on sensory data):
12 - quality (score between 0 and 10)'''
```

Out[1]:
```
'\nWine quality dataset 11 input features and 1 output feature\n\n1 - fixed a
cidity\n2 - volatile acidity\n3 - citric acid\n4 - residual sugar\n5 - chlori
des\n6 - free sulfur dioxide\n7 - total sulfur dioxide\n8 - density\n9 - pH\n
10 - sulphates\n11 - alcohol\nOutput variable (based on sensory data):\n12 -
quality (score between 0 and 10)'
```

import modules for numpy

In [2]:
```

```

In [3]:
```
wines = np.genfromtxt("winequality-red.csv", delimiter=";", skip_header=1)
```

What is its size?

In [4]:

```
```

Out[4]: (1599, 12)

How many wine data rows here?

In [5]:

```
```

Out[5]: 1599

How many wine data columns here?

In [6]:

```
```

Out[6]: 12

How many dimensions?

In [7]:

```
```

Out[7]: 2

What is the type of wines?

In [8]:

```
```

Out[8]: numpy.ndarray

What is the data type of wines data?

In [9]:

```
```

Out[9]: dtype('float64')

Show top 5 rows

In [10]:

```
```

What is the value at 3rd row, 4th column of wine data?

In [11]:

Out[11]:
```
2.3
```

Select first 3 items in 4th column

In [12]:

Out[12]:
```
array([1.9, 2.6, 2.3])
```

Show 1st column

In [13]:

Out[13]:
```
array([7.4, 7.8, 7.8, ..., 6.3, 5.9, 6. ])
```

Show 2nd row

In [14]:

Out[14]:
```
array([ 7.8   ,  0.88  ,  0.    ,  2.6   ,  0.098 , 25.    , 67.    ,
        0.9968,  3.2   ,  0.68  ,  9.8   ,  5.    ])
```

Select items from rows 1 to 3 and 5th column

In [15]:

Out[15]:
```
array([0.098, 0.092, 0.075])
```

Select entire array

In [16]:

Out[16]:
```
array([[ 7.4  ,  0.7  ,  0.   , ...,  0.56 ,  9.4  ,  5.   ],
       [ 7.8  ,  0.88 ,  0.   , ...,  0.68 ,  9.8  ,  5.   ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  0.65 ,  9.8  ,  5.   ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  0.75 , 11.   ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ],
       [ 6.   ,  0.31 ,  0.47 , ...,  0.66 , 11.   ,  6.   ]])
```

Change 1st value in wines to 100

In [17]:
```
# show actual value
```
Out[17]:
```
7.4
```

In [18]:
```
# update
```

In [19]:
```
# show updated value
```
Out[19]:
```
100.0
```

change it back to 7.4 and print

In [20]:
```
```

1-Dimensional Numpy Arrays

Select 4th row all column values

In [21]:
```
```

display its value

In [22]:
```
```
Out[22]:
```
array([11.2  ,  0.28 ,  0.56 ,  1.9  ,  0.075, 17.   , 60.   ,  0.998,
        3.16 ,  0.58 ,  9.8  ,  6.   ])
```

show 2nd value

In [23]:
```
```
Out[23]:
```
0.28
```

Convert wine data to integer values and show it

In [24]:
```
#convert to int
```

Out[24]:
```
array([[ 7,  0,  0, ...,  0,  9,  5],
       [ 7,  0,  0, ...,  0,  9,  5],
       [ 7,  0,  0, ...,  0,  9,  5],
       ...,
       [ 6,  0,  0, ...,  0, 11,  6],
       [ 5,  0,  0, ...,  0, 10,  5],
       [ 6,  0,  0, ...,  0, 11,  6]])
```

Vectorization Operations

Increase wine quality score (output variable) by 10

In [25]:
```
# check values first
```

Out[25]:
```
array([5., 5., 5., ..., 6., 5., 6.])
```

Increase by 10

In [26]:
```
```

Display update score

In [28]:
```
```

Out[28]:
```
array([15., 15., 15., ..., 16., 15., 16.])
```

Multiply alcohol of all wine data by 3 times

In [29]:
```
```

Show updated alcohol column

In [30]:
```
```

Out[30]:
```
array([28.2, 29.4, 29.4, ..., 33. , 30.6, 33. ])
```

Add quality column by itselt

In [31]:
```
# It will produce a new array
```

Out[31]:
```
array([30., 30., 30., ..., 32., 30., 32.])
```

Multiply alcohol and wine quality columns. It will perform element wise multiplication

In [32]:

Out[32]:
```
array([423., 441., 441., ..., 528., 459., 528.])
```

Broadcasting

Add every row of wines data with a random array of values

In [33]:

Show rand_array

In [34]:

Out[34]:
```
array([0.72587682, 0.9600024 , 0.17236312, 0.56655827, 0.20321856,
       0.47046892, 0.88980548, 0.40083145, 0.97884246, 0.2112331 ,
       0.33194581, 0.72612594])
```

add wines and rand_array

In [35]:

Out[35]:
```
array([[ 8.12587682,  1.6600024 ,  0.17236312, ...,  0.7712331 ,
        28.53194581, 15.72612594],
       [ 8.52587682,  1.8400024 ,  0.17236312, ...,  0.8912331 ,
        29.73194581, 15.72612594],
       [ 8.52587682,  1.7200024 ,  0.21236312, ...,  0.8612331 ,
        29.73194581, 15.72612594],
       ...,
       [ 7.02587682,  1.4700024 ,  0.30236312, ...,  0.9612331 ,
        33.33194581, 16.72612594],
       [ 6.62587682,  1.6050024 ,  0.29236312, ...,  0.9212331 ,
        30.93194581, 15.72612594],
       [ 6.72587682,  1.2700024 ,  0.64236312, ...,  0.8712331 ,
        33.33194581, 16.72612594]])
```

In [ ]:

Department of Data Science - Data and Visual Analytics Lab

Lab2. Red Wine Quality Data Analysis using NumPy Part-II

Objectives

In this lab, you will continue to work on analyzing red wine quality dataset.

How To Use This Notebook

For each question, you should write NumPy statements in the "In[]" Cell and the expected output "Out[]" is already shown just below all In[] cells.

Import necessary modules

In [ ]:

```
```

In [2]:

```
wines = np.genfromtxt("winequality-red.csv", delimiter=";", skip_header=1)
```

NumPy Aggregation Methods

Find sum of all residual sugar values

In [3]:

```
```

Out[3]:

```
4059.55
```

Find sums of every feature value. There are 12 features altogether

In [4]:

```
```

Out[4]:

```
array([13303.1    ,   843.985  ,   433.29   ,  4059.55   ,   139.859  ,
        25384.     , 74302.     ,  1593.79794,  5294.47   ,  1052.38   ,
        16666.35   ,  9012.     ])
```

Find sum of every row

In [5]:

```
array([ 74.5438 , 123.0548 ,  99.699  , ..., 100.48174, 105.21547,
        92.49249])
```

What is its size?

In [6]:

```
(1599,)
```

What is the maximum residual sugar value in red wines data?

In [7]:

```
# convert sugar value into int data type first
```

```
array([1, 2, 2, ..., 2, 2, 3])
```

find its maximum residual sugar value

In [8]:

```
15
```

What is the minimum residual sugar value in red wines data?

In [9]:

```
0
```

What is the average residual sugar value in red wines data?

In [10]:

```
2.53880550343965
```

What is 25 percentile residual sugar value?

In [11]:

```
1.9
```

What is 75 percentile residual sugar value?

In [12]:

```

```

Out[12]:

```
2.6
```

Find the average of each feature value

In [13]:

```

```

Out[13]:

```
array([ 8.31963727,  0.52782051,  0.27097561,  2.5388055 ,  0.08746654,
       15.87492183, 46.46779237,  0.99674668,  3.3111132 ,  0.65814884,
       10.42298311,  5.63602251])
```

NumPy Array Comparisons

Show all wines with quality > 5

In [14]:

```

```

Out[14]:

```
array([False, False, False, ...,  True, False,  True])
```

Show all wines with quality > 7

In [15]:

```

```

Out[15]:

```
array([False, False, False, ..., False, False, False])
```

check if any wines value is True for the condition quality > 7

In [16]:

```

```

Out[16]:

```
True
```

Show first 3 rows where wine quality > 7, call it high_quality

In [17]:

```
high_quality =
```

In [18]:

```
high_quality
```

Out[18]:

```
array([False, False, False, ..., False, False, False])
```

Show only top 3 rows and all columns of high_quality wines data

In [19]:

Out[19]:
```
array([[7.900e+00, 3.500e-01, 4.600e-01, 3.600e+00, 7.800e-02, 1.500e+01,
        3.700e+01, 9.973e-01, 3.350e+00, 8.600e-01, 1.280e+01, 8.000e+00],
       [1.030e+01, 3.200e-01, 4.500e-01, 6.400e+00, 7.300e-02, 5.000e+00,
        1.300e+01, 9.976e-01, 3.230e+00, 8.200e-01, 1.260e+01, 8.000e+00],
       [5.600e+00, 8.500e-01, 5.000e-02, 1.400e+00, 4.500e-02, 1.200e+01,
        8.800e+01, 9.924e-01, 3.560e+00, 8.200e-01, 1.290e+01, 8.000e+00]])
```

Show wines with a lot of alcohol > 10 and high wine quality > 7

In [20]:

show only alcohol and wine quality columns

In [21]:

Out[21]:
```
array([[12.8,  8. ],
       [12.6,  8. ],
       [12.9,  8. ],
       [13.4,  8. ],
       [11.7,  8. ],
       [11. ,  8. ],
       [11. ,  8. ],
       [14. ,  8. ],
       [12.7,  8. ],
       [12.5,  8. ],
       [11.8,  8. ],
       [13.1,  8. ],
       [11.7,  8. ],
       [14. ,  8. ],
       [11.3,  8. ],
       [11.4,  8. ]])
```

Combining NumPy Arrays

Combine red wine and white wine data

Open white wine dataset

In [22]:
```
white_wines = np.genfromtxt("winequality-white.csv", delimiter=";", skip_heade
r=1)
```

Show size of white_wines

In [ ]:
```

```

combine wines and white_wines data frames using vstack and call it all_wines

In [23]:
```

```

In [24]:
```
# what is size of all_wines?
```

Out[24]: (6497, 12)

In [ ]:
```

```

Combine wines and white_wines data frames using concatenate method

In [25]:
```

```

In [26]:
```
# size of data2
```

Matrix Operations and Reshape

Find Transpose of wines and print its size

In [27]:
```

```

Out[27]: (12, 1599)

Convert wines data into 1D array

In [28]:
```

```

Out[28]: array([7.4 , 0.7 , 0. , ..., 0.66, 11. , 6.])

In [29]:
```
# show size
```

Out[29]: (19188,)

Reshape second row of wines into a 2-dimensional array with 2 rows and 6 columns

In [30]:

Out[30]:
```
array([[ 7.8   ,  0.88  ,  0.    ,  2.6   ,  0.098 , 25.    ],
       [67.    ,  0.9968,  3.2   ,  0.68  ,  9.8   ,  5.    ]])
```

Sort alcohol column Ascending Order

In [31]:
```
sorted_alcohol =
```

In [32]:
```
sorted_alcohol
```

Out[32]:
```
array([ 8.4,  8.4,  8.5, ..., 14. , 14. , 14.9])
```

Make sorting to take place in-place

In [33]:
```
# In-place sorting
```

Show top 10 rows

In [34]:

Out[34]:
```
array([ 8.4,  8.4,  8.5, ..., 14. , 14. , 14.9])
```

Sort alcohol column Descending Order

In [35]:
```
sorted_alcohol_desc =
```

In [36]:
```
sorted_alcohol_desc
```

Out[36]:
```
array([14.9, 14. , 14. , ...,  8.5,  8.4,  8.4])
```

Will original data be modified?. Check top 10 rows

In [37]:

Out[37]:
```
array([ 8.4,  8.4,  8.5, ..., 14. , 14. , 14.9])
```

In [ ]:

Department of Data Science - Data and Visual Analytics Lab

Lab3. Pandas Indexing and Selection

Objectives

In this lab

```
- you will learn how to create Series object and Dataframe object.
- Then, you will learn to access elements using index and select rows and columns u
sing positions and column labels.
- You will finally learn aggregate functions and math operators in Pandas
```

Simple Series and DataFrames

Import necessary modules

In [1]:

```

```

Create a Series to store Temperature values for 1 week

In [2]:

```
temperature_trichy = pd.Series([40.2, 39.8, 36.3, 39.1, 41.3, 32.9, 36.6])
```

show temperature values

In [3]:

```

```

Out[3]:

```
0    40.2
1    39.8
2    36.3
3    39.1
4    41.3
5    32.9
6    36.6
dtype: float64
```

What is the weather on 2nd day?

In [4]:

Out[4]:
```
39.8
```

Find all days and temperatures where temperature over 40.0 degree Celsius

In [5]:

Out[5]:
```
0    40.2
4    41.3
dtype: float64
```

Find only day, not temperature where temperature over 40.0 degree Celsius

In [6]:

Out[6]:
```
Int64Index([0, 4], dtype='int64')
```

Create a Dataframe for student details from List

In [7]:
```
students = [['DS01', 'Rex', '1msc'], ['DS02', 'peter', '2msc'], ['CS01', 'ann'
, '3bsc']]

df_stud = pd.DataFrame(students, columns=['rollno', 'name', 'class'])  # row i
ndex automatically generated
```

show df_stud dataframe

In [8]:

Out[8]:

	rollno	name	class
0	DS01	Rex	1msc
1	DS02	peter	2msc
2	CS01	ann	3bsc

Display all column names of df_stud

```
In [9]:
```

```
Out[9]: Index(['rollno', 'name', 'class'], dtype='object')
```

Add a new column "address" with values ['Delhi', 'Bangalore', 'Chennai'] to df_stud

```
In [10]:
```

```
In [11]: df_stud
```

Out[11]:

	rollno	name	class	address
0	DS01	Rex	1msc	Delhi
1	DS02	peter	2msc	Bangalore
2	CS01	ann	3bsc	Chennai

Create a Dataframe for Phone book from Dictionary

```
In [12]: phonebook = {'rex':[9942002764, 'rex@abc.com'], 'sam':[9932176542, 'sam@xyz.co
m'], 'peter':[9865323645, 'ann@bhc.com']}
df_phonebook = pd.DataFrame.from_dict(phonebook, orient='index', columns=['mob
ile', 'email'])
```

Display df_phonebook

```
In [ ]:
```

Exploratory Data Analysis on Video Game Review Dataset

Import ign.csv dataset

```
In [13]: reviews = pd.read_csv("ign.csv")
```

Show top-5 rows

In [14]:

Out[14]:

	Unnamed: 0	score_phrase	title	url	platform	score	genre	ed
0	0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	
1	1	Amazing	LittleBigPlanet PS Vita -- Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	
2	2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	
3	3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	
4	4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	

Show bottom 3 rows

In [15]:

Out[15]:

	Unnamed: 0	score_phrase	title	url	platform	score	genre	edito
18622	18622	Mediocre	Star Ocean: Integrity and Faithlessness	/games/star-ocean-5/ps4-20035681	PlayStation 4	5.8	RPG	
18623	18623	Masterpiece	Inside	/games/inside-playdead/xbox-one-121435	Xbox One	10.0	Adventure	
18624	18624	Masterpiece	Inside	/games/inside-playdead/pc-20055740	PC	10.0	Adventure	

How many rows and columns here?

In [16]:

Out[16]: (18625, 11)

What are the datatypes?

In [17]:

Out[17]:
```
Unnamed: 0           int64
score_phrase        object
title               object
url                 object
platform            object
score              float64
genre               object
editors_choice      object
release_year         int64
release_month        int64
release_day          int64
dtype: object
```

Selecting Columns

Select a single column, say title and print head

In [18]:

Out[18]:
```
18620                   Tokyo Mirage Sessions #FE
18621            LEGO Star Wars: The Force Awakens
18622      Star Ocean: Integrity and Faithlessness
18623                                       Inside
18624                                       Inside
Name: title, dtype: object
```

Select multiple columns, title and genre and print head

In [19]:

Out[19]:

	title	genre
0	LittleBigPlanet PS Vita	Platformer
1	LittleBigPlanet PS Vita -- Marvel Super Hero E...	Platformer
2	Splice: Tree of Life	Puzzle
3	NHL 13	Sports
4	NHL 13	Sports
5	Total War Battles: Shogun	Strategy
6	Double Dragon: Neon	Fighting
7	Guild Wars 2	RPG
8	Double Dragon: Neon	Fighting
9	Total War Battles: Shogun	Strategy

Selection using Positions

Select top-5 rows and all columns, same as head() using iloc

In [20]:

Out[20]:

	Unnamed: 0	score_phrase	title	url	platform	score	genre	ed
0	0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	
1	1	Amazing	LittleBigPlanet PS Vita -- Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	
2	2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	
3	3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	
4	4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	

Select rows from position 5 onwards, and columns from position 5 onwards.

In [ ]:

Select the first column, and all of the rows for the column

In [ ]:

the 10th row, and all of the columns for that row.

In [ ]:

First column is not useful. So remove it

In [21]:

Out[21]:

	score_phrase	title	url	platform	score	genre	editors_choice
0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y
1	Amazing	LittleBigPlanet PS Vita -- Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y
2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N
3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N
4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N

Selection using Row and Column Labels

We have already created students dataframe as below. Let us access name column with loc()

In [22]:

```
students = [['DS01', 'Rex', '1msc'], ['DS02', 'peter', '2msc'], ['CS01', 'ann'
, '3bsc']]
df_stud = pd.DataFrame(students, columns=['rollno', 'name', 'class'])  # row i
ndex automatically generated
```

In [23]:

```
df_stud
```

Out[23]:

	rollno	name	class
0	DS01	Rex	1msc
1	DS02	peter	2msc
2	CS01	ann	3bsc

Print all names using loc

In [24]:

Out[24]:

```
0      Rex
1    peter
2      ann
Name: name, dtype: object
```

Let us come back to our reviews. Display the first five rows of reviews using the loc method

In [25]:

Out[25]:

	score_phrase	title	url	platform	score	genre	editors_choice
0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y
1	Amazing	LittleBigPlanet PS Vita -- Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y
2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N
3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N
4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N
5	Good	Total War Battles: Shogun	/games/total-war-battles-shogun/mac-142565	Macintosh	7.0	Strategy	N

Select score_phrase column using loc and print head

In [26]:

Out[26]:

```
0    Amazing
1    Amazing
2      Great
3      Great
4      Great
Name: score_phrase, dtype: object
```

Print top 10 values of column label "score_phrase"

In [27]:

Out[27]:
```
0    Amazing
1    Amazing
2      Great
3      Great
4      Great
5       Good
6      Awful
7    Amazing
8      Awful
9       Good
Name: score_phrase, dtype: object
```

Select from reviews of rows from 5 to 15

In [28]:
```
some_reviews =
```

print top 5 rows from some_reviews

In [29]:

Out[29]:

	score_phrase	title	url	platform	score	genre	editors_choice	release_yea
5	Good	Total War Battles: Shogun	/games/total-war-battles-shogun/mac-142565	Macintosh	7.0	Strategy	N	201
6	Awful	Double Dragon: Neon	/games/double-dragon-neon/xbox-360-131320	Xbox 360	3.0	Fighting	N	201
7	Amazing	Guild Wars 2	/games/guild-wars-2/pc-896298	PC	9.0	RPG	Y	201
8	Awful	Double Dragon: Neon	/games/double-dragon-neon/ps3-131321	PlayStation 3	3.0	Fighting	N	201
9	Good	Total War Battles: Shogun	/games/total-war-battles-shogun/pc-142564	PC	7.0	Strategy	N	201

Select scores of first 3 rows some_reviews

In [30]:

Out[30]:
```
5    7.0
6    3.0
7    9.0
Name: score, dtype: float64
```

Select "score", "genre", and "release_year" columns from reviews dataframe and print head

In [31]:

Out[31]:

	score	genre	release_year
0	9.0	Platformer	2012
1	9.0	Platformer	2012
2	8.5	Puzzle	2012
3	8.5	Sports	2012
4	8.5	Sports	2012

What is the datatype of "score" column?

In [32]:

Out[32]:
```
pandas.core.series.Series
```

Aggregate Columns

Find average value of score column in reviews dataframe

In [33]:

Out[33]:
```
6.950459060402666
```

Find average value of all numeric columns

In [34]:

Out[34]:
```
score               6.950459
release_year     2006.515329
release_month       7.138470
release_day        15.603866
dtype: float64
```

Find average value for each numeric column

In [35]:

Out[35]:
```
score               6.950459
release_year     2006.515329
release_month       7.138470
release_day        15.603866
dtype: float64
```

Find average value for each row containing numeric values and print head

In [36]:

Out[36]:
```
0    510.500
1    510.500
2    510.375
3    510.125
4    510.125
dtype: float64
```

Find lowest, highest, median, standard deviation of score column of reviews dataframe

show median of "score" column of reviews dataframe

In [37]:

Out[37]:
```
7.3
```

show minimum of "score" column of reviews dataframe

In [38]:

Out[38]:
```
0.5
```

show maximum of "score" column of reviews dataframe

In [39]:

Out[39]:
```
10.0
```

show standard deviation of "score" column of reviews dataframe

In [40]:

Out[40]: 1.7117358608045874

How many non-null values in "score" column of reviews dataframe?

In [41]:

Out[41]: 18625

Show the summary of reviews dataframe

In [42]:

Out[42]:

	score	release_year	release_month	release_day
count	18625.000000	18625.000000	18625.00000	18625.000000
mean	6.950459	2006.515329	7.13847	15.603866
std	1.711736	4.587529	3.47671	8.690128
min	0.500000	1970.000000	1.00000	1.000000
25%	6.000000	2003.000000	4.00000	8.000000
50%	7.300000	2007.000000	8.00000	16.000000
75%	8.200000	2010.000000	10.00000	23.000000
max	10.000000	2016.000000	12.00000	31.000000

Check if review score has any correlation with other columns of reviews

In [43]:

Out[43]:

	score	release_year	release_month	release_day
score	1.000000	0.062716	0.007632	0.020079
release_year	0.062716	1.000000	-0.115515	0.016867
release_month	0.007632	-0.115515	1.000000	-0.067964
release_day	0.020079	0.016867	-0.067964	1.000000

Review score has no correlation with other features. So, release timing doesn't linearly relate to review score

Math Operations on DF columns

Divide the values of "score" column in reviews dataframe by 2. There will be too many values, so just print head

In [44]:

Out[44]:
```
0    4.50
1    4.50
2    4.25
3    4.25
4    4.25
Name: score, dtype: float64
```

Boolean Indexing in Pandas

Select all video games whose review score > 7, call it score_filter

In [45]:
```
score_filter =
```

Print head of score_filter

In [46]:

Out[46]:
```
0    True
1    True
2    True
3    True
4    True
Name: score, dtype: bool
```

Select all rows for score_filter column and print its head

In [47]:
```
filtered_reviews =
```

Out[47]:

	score_phrase	title	url	platform	score	genre	editors_choice
0	Amazing	LittleBigPlanet PS Vita	/games/littlebigplanet-vita/vita-98907	PlayStation Vita	9.0	Platformer	Y
1	Amazing	LittleBigPlanet PS Vita -- Marvel Super Hero E...	/games/littlebigplanet-ps-vita-marvel-super-he...	PlayStation Vita	9.0	Platformer	Y
2	Great	Splice: Tree of Life	/games/splice/ipad-141070	iPad	8.5	Puzzle	N
3	Great	NHL 13	/games/nhl-13/xbox-360-128182	Xbox 360	8.5	Sports	N
4	Great	NHL 13	/games/nhl-13/ps3-128181	PlayStation 3	8.5	Sports	N

Show the size of filtered_reviews

In [48]:

Out[48]:
```
(9800, 10)
```

Show top 10 "title" from filtered_reviews

In [49]:

Out[49]:
```
0                             LittleBigPlanet PS Vita
1     LittleBigPlanet PS Vita -- Marvel Super Hero E...
2                                Splice: Tree of Life
3                                              NHL 13
4                                              NHL 13
7                                        Guild Wars 2
10                          Tekken Tag Tournament 2
11                          Tekken Tag Tournament 2
13                                 Mark of the Ninja
14                                 Mark of the Ninja
Name: title, dtype: object
```

Find games released for the Xbox One platform that have a score of more than 7

First create a filter, called xbox_one_filter for the conditions

In [50]:
```
xbox_one_filter =
```

Select those rows from reviews of xbox_one_filter and print head

In [51]:
```
filtered_reviews2 =




#show top 5 rows of filtered_reviews2
```

Out[51]:

	score_phrase	title	url	platform	score	genre	editors_choice	releas
17137	Amazing	Gone Home	/games/gone-home/xbox-one-20014361	Xbox One	9.5	Simulation	Y	
17197	Amazing	Rayman Legends	/games/rayman-legends/xbox-one-20008449	Xbox One	9.5	Platformer	Y	
17295	Amazing	LEGO Marvel Super Heroes	/games/lego-marvel-super-heroes/xbox-one-20000826	Xbox One	9.0	Action	Y	
17313	Great	Dead Rising 3	/games/dead-rising-3/xbox-one-124306	Xbox One	8.3	Action	N	
17317	Great	Killer Instinct	/games/killer-instinct-2013/xbox-one-20000538	Xbox One	8.4	Fighting	N	

In [52]:
```
# What is the size of filtered_reviews2
```

Out[52]: (140, 10)

Select all video games which are 'Action' genre

In [53]:
```
action_reviews =
```

In [54]:
```
action_reviews.head()
```

Out[54]:

	score_phrase	title	url	platform	score	genre	editors_choice	release_ye
17	Great	Avengers Initiative	/games/avengers-initiative/iphone-141579	iPhone	8.0	Action	N	20
34	Good	War of the Roses	/games/war-of-the-roses-140577/pc-115849	PC	7.3	Action	N	20
45	Amazing	Bad Piggies	/games/bad-piggies/iphone-141455	iPhone	9.2	Action	Y	20
49	Okay	Demon's Score	/games/demons-score/iphone-118050	iPhone	6.9	Action	N	20
69	Great	Hotline Miami	/games/hotline-miami/pc-139657	PC	8.8	Action	Y	20

In [55]:
```
What is the size of action_reviews?
```

Out[55]:
```
(3797, 10)
```

Plot Review Ratings of two Play Stations and Compare Which one has more ratings?

Now that we know how to filter, we can create plots to observe the review distribution for the Xbox One vs the review distribution for the PlayStation 4. This will help us figure out which console has better games.

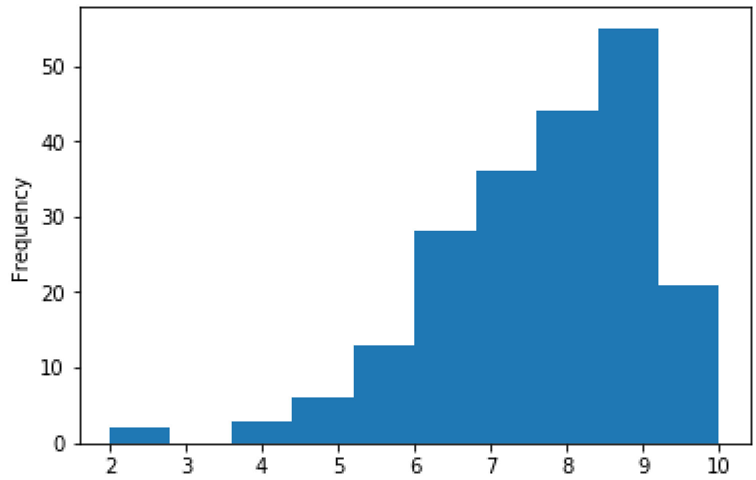
We can do this via a histogram, which will plot the frequencies for different score ranges.

Plot Histogram for the frequencies of different score ranges of Xbox One platform

In [56]:
```
# Import plotting libraries
```

```python
In [57]: # Plot the following histogram of score values for Xbox One platform
```

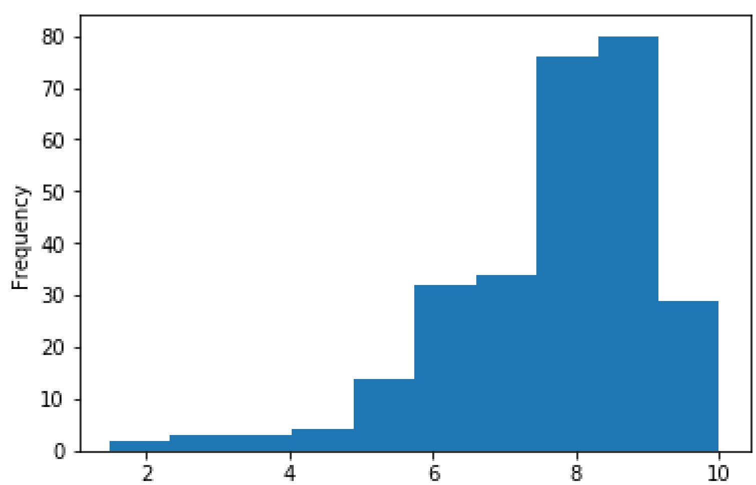
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x25161f78c88>



Plot Histogram for Frequencies of the scores of Play Station4 platform

```python
In [58]:
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x2516407e7b8>



Therefore, it appears from our histograms that the PlayStation4 has many more highly rated games than the Xbox One.

```python
In [ ]:
```

Department of Data Science - Data and Visual Analytics Lab

Lab4. Pandas Grouping and Aggregation

Objectives

In this lab, you will learn how to

```
- apply functions to Series and Dataframe
- group data in Pandas
- aggregate values in groups
- plot the results of aggregation
- aggregate multiple columns and multiple functions
```

You will explore what Americans typically eat for Thanksgiving dinner. The dataset contains 1058 online survey responses collected by FiveThirtyEight.

Each survey respondent was asked questions about what they typically eat for Thanksgiving, along with some demographic questions, like their gender, income, and location.

This dataset will allow us to discover regional and income-based patterns in what Americans eat for Thanksgiving dinner.
Now, we will compute group summary statistics, discover patterns, and slice up the data in various ways.

Import necessary modules

In [1]:
```

```

In [ ]:
```
data = pd.read_csv("thanksgiving-2015-poll-data.csv", encoding="Latin-1")
```

In [2]:
```
# Print top 5 rows from data
```
Out[2]:

	RespondentID	Do you celebrate Thanksgiving?	What is typically the main dish at your Thanksgiving dinner?	What is typically the main dish at your Thanksgiving dinner? - Other (please specify)	How is the main dish typically cooked?	How is the main dish typically cooked? - Other (please specify)	What kind stuffing/dressi do you typica hav
0	4337954960	Yes	Turkey	NaN	Baked	NaN	Bread-bas
1	4337951949	Yes	Turkey	NaN	Baked	NaN	Bread-bas
2	4337935621	Yes	Turkey	NaN	Roasted	NaN	Rice-bas
3	4337933040	Yes	Turkey	NaN	Baked	NaN	Bread-bas
4	4337931983	Yes	Tofurkey	NaN	Baked	NaN	Bread-bas

5 rows × 65 columns

In [3]:
```
# what is the size?
```
Out[3]:
```
(1058, 65)
```

As you can see above, the data has 65 columns of mostly categorical data. For example, the first column appears to allow for Yes and No responses only. Let’s verify by using the pandas.Series.unique method to see what unique values are in the Do you celebrate Thanksgiving? column of data.

What are unique values of "Do you celebrate Thanksgiving?" column?

In [4]:
```
```
Out[4]:
```
array(['Yes', 'No'], dtype=object)
```

View all column names (top 5)

In [5]:

```
Out[5]: Index(['RespondentID', 'Do you celebrate Thanksgiving?',
       'What is typically the main dish at your Thanksgiving dinner?',
       'What is typically the main dish at your Thanksgiving dinner? - Other
(please specify)',
       'How is the main dish typically cooked?'],
      dtype='object')
```

Apply function to Series

DATA CLEANING - Now, let us transform gender to numeric value.

We'll assign 0 to Male, and 1 to Female. Before we dive into transforming the values, let's confirm that the values in the column are either Male or Female. We can use the pandas.Series.value_counts method to help us with this. We'll pass the dropna=False keyword argument to also count missing values.

How many male, female and NaN in "What is your gender?" column

In [6]:

```
Out[6]: Female    544
        Male      481
        NaN        33
        Name: What is your gender?, dtype: int64
```

Yes, they are female, male or nan

Let apply a user defined function to each value in the What is your gender? column to transform Male to 0 and female to 1

In [7]:

```
import math

def gender_code(gender_string):
    if isinstance(gender_string, float) and math.isnan(gender_string):
        return gender_string
    return int(gender_string == "Female")
```

Apply gender_code() to What is your gender? column

Let us apply this function to every row of What is your gender? column. It is something like automatic looping. Create a new column 'gender' and put it there

In [8]:

Now, count male and females as 0s and 1s. How many in "gender" column?

In [9]:

Out[9]:
```
1.0    544
0.0    481
NaN     33
Name: gender, dtype: int64
```

Applying functions to DataFrames

The apply method will work across each column in the DataFrame. If we pass the axis=1 keyword argument, it will work across each row.

Check the data type of each column in data using a lambda function. Just visualize data types of first 5 columns

In [10]:

Out[10]:
```
RespondentID
object
Do you celebrate Thanksgiving?
object
What is typically the main dish at your Thanksgiving dinner?
object
What is typically the main dish at your Thanksgiving dinner? - Other (please
specify)    object
How is the main dish typically cooked?
object
dtype: object
```

DATA CLEANING - Let us clean up Income column

We need to convert string values representing income in **"How much total combined money did all members of your HOUSEHOLD earn last year"** column into numeric values. Check the unique values first

In [11]:

```

```

Out[11]:

```
$25,000 to $49,999      180
Prefer not to answer    136
$50,000 to $74,999      135
$75,000 to $99,999      133
$100,000 to $124,999    111
$200,000 and up          80
$10,000 to $24,999       68
$0 to $9,999             66
$125,000 to $149,999     49
$150,000 to $174,999     40
NaN                      33
$175,000 to $199,999     27
Name: How much total combined money did all members of your HOUSEHOLD earn la
st year?, dtype: int64
```

Looking at this, there are 4 different patterns for the values in the column: X to Y — an example is $25,000 to$49,999. We can convert this to a numeric value by extracting the numbers and averaging them. NaN We'll preserve NaN values, and not convert them at all. X and up — an example is $200,000 and up. We can convert this to a numeric value by extracting the number. Prefer not to answer We'll turn this into an NaN value.

In [12]:

```python
import numpy as np

def clean_income(value):

    if value == "$200,000 and up":
        return 200000
    elif value == "Prefer not to answer":
        return np.nan
    elif isinstance(value, float) and math.isnan(value):
        return np.nan

    value = value.replace("$", "").replace(",", "")
    income_high, income_low = value.split(" to ")

    return (int(income_high) + int(income_low)) / 2
```

Now apply this function to the "How much total combined money did all members of your HOUSEHOLD earn last year?" column and put it in new column "income"

In [ ]:

```python
data["income"] =
```

In [13]:

```python
data["income"].head()
```

Out[13]:

```
0     87499.5
1     62499.5
2      4999.5
3    200000.0
4    112499.5
Name: income, dtype: float64
```

Grouping Data with Pandas

Who earn more income?

Suppose, we want to find who earn more income?. Is it People eating homemade sauce or people eating canned sauce during the Thanksgiving Day?

Check unique values in column, "What type of cranberry saucedo you typically have?" first.

In [14]:

```
```

Out[14]:

```
Canned                     502
Homemade                   301
None                       146
Other (please specify)      25
Name: What type of cranberry saucedo you typically have?, dtype: int64
```

We can now filter data to get two DataFrames, namely, homemade_df & canned_df, that only contain rows where the What type of cranberry saucedo you typically have? is Canned or Homemade, respectively

Create a datafrme by filtering values "Homemade"

In [15]:

```
homemade_df =
```

Create another datafrme by filtering values "Canned"

In [16]:

```
canned_df =
```

Now print mean income of homemade_df and canned_df for these two groups of people

In [17]:

```
```

```
94878.1072874494
83823.40340909091
```

Conclusion: Wow, great. We can understand from these values that **people who eat home made cranberry sauce earn more income** that the other group.

In [ ]:

```

```

Use groupby() and aggregate() to find out "Who earn more income?"

Split dataset based on "What type of cranberry saucedo you typically have?" column automatically into groups based on unique values

In [18]:

```
grouped = 

grouped
```

Out[18]:

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000197B706D7F0>
```

List out all groups that are created by groupby()

In [19]:

```

```

Out[19]:

```
{'Canned': Int64Index([   4,    6,    8,   11,   12,   15,   18,   19,   26,
27,
            ...
            1040, 1041, 1042, 1044, 1045, 1046, 1047, 1051, 1054, 1057],
           dtype='int64', length=502),
 'Homemade': Int64Index([   2,    3,    5,    7,   13,   14,   16,   20,   2
1,   23,
            ...
            1016, 1017, 1025, 1027, 1030, 1034, 1048, 1049, 1053, 1056],
           dtype='int64', length=301),
 'None': Int64Index([   0,   17,   24,   29,   34,   36,   40,   47,   49,
51,
            ...
             980,  981,  997, 1015, 1018, 1031, 1037, 1043, 1050, 1055],
           dtype='int64', length=146),
 'Other (please specify)': Int64Index([   1,    9,  154,  216,  221,  233,  2
49,  265,  301,  336,  380,
             435,  444,  447,  513,  550,  749,  750,  784,  807,  860,  87
2,
             905, 1000, 1007],
           dtype='int64')}
```

In [20]:

```
grouped.size()
```

Out[20]:

```
What type of cranberry saucedo you typically have?
Canned                        502
Homemade                      301
None                          146
Other (please specify)         25
dtype: int64
```

```
In [21]: for name, group in grouped:
             print(name)
             print(group.shape)
             print(type(group))
```

```
Canned
(502, 67)
<class 'pandas.core.frame.DataFrame'>
Homemade
(301, 67)
<class 'pandas.core.frame.DataFrame'>
None
(146, 67)
<class 'pandas.core.frame.DataFrame'>
Other (please specify)
(25, 67)
<class 'pandas.core.frame.DataFrame'>
```

Here each group is a DataFrame, and you can use any normal DataFrame methods on it. We can also extract a single column from a group. This will allow us to perform further computations just on that specific column:

```
In [22]: grouped["income"]
```

```
Out[22]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x00000197B707D4E0>
```

```
In [23]: grouped["income"].size()
```

```
Out[23]: What type of cranberry saucedo you typically have?
         Canned                      502
         Homemade                    301
         None                        146
         Other (please specify)       25
         Name: income, dtype: int64
```

Aggregating values in groups

Spliting data into groups will not be sufficient. Real power comes when we can apply computation on each group.

Now, find out average income

We could find the average income for people who served each type of cranberry sauce. Extract income column from grouped DF and fine mean value for each group

In [24]:

Out[24]:
```
What type of cranberry saucedo you typically have?
Canned                        83823.403409
Homemade                      94878.107287
None                          78886.084034
Other (please specify)        86629.978261
Name: income, dtype: float64
```

If you want to consider all numberic attributes and find the mean for each group for every column in data, you can do as below.

In [25]:

Out[25]:

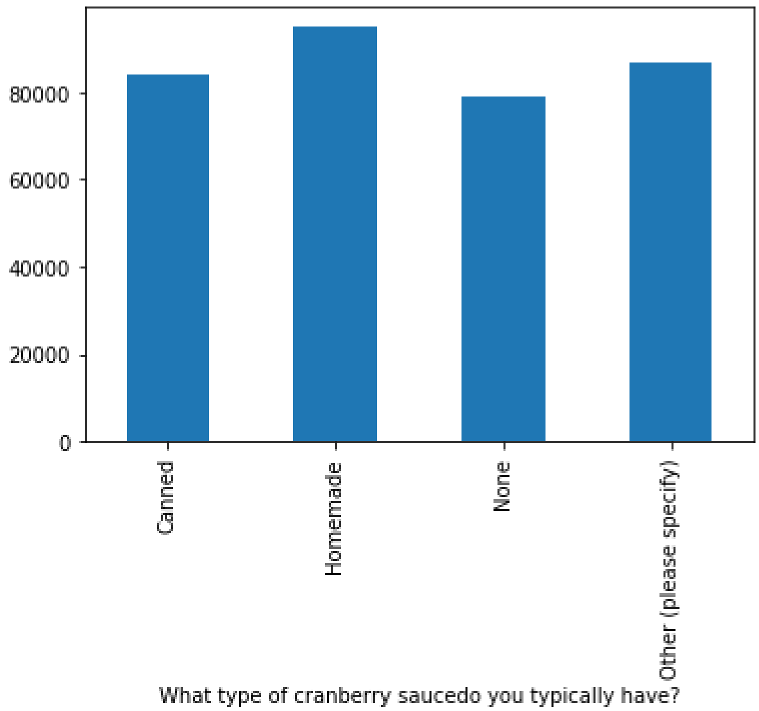
	RespondentID	gender	income
What type of cranberry saucedo you typically have?			
Canned	4.336699e+09	0.552846	83823.403409
Homemade	4.336792e+09	0.533101	94878.107287
None	4.336765e+09	0.517483	78886.084034
Other (please specify)	4.336763e+09	0.640000	86629.978261

Plotting the results of aggregation

What is the average income of each category?

In [26]:

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x197b70886d8>



Aggregating with multiple columns

Find the average income of people who eat Homemade cranberry sauce and Tofurkey

We need to apply groupby on two columns "What type of cranberry saucedo you typically have?" and "What is typically the main dish at your Thanksgiving dinner?"

In [27]:

Out[27]:

What type of cranberry saucedo you typically have?	What is typically the main dish at your Thanksgiving dinner?	RespondentID	gender	income
Canned	Chicken	4.336354e+09	0.333333	80999.600000
	Ham/Pork	4.336757e+09	0.642857	77499.535714
	I don't know	4.335987e+09	0.000000	4999.500000
	Other (please specify)	4.336682e+09	1.000000	53213.785714
	Roast beef	4.336254e+09	0.571429	25499.500000
	Tofurkey	4.337157e+09	0.714286	100713.857143
	Turkey	4.336705e+09	0.544444	85242.682045
Homemade	Chicken	4.336540e+09	0.750000	19999.500000
	Ham/Pork	4.337253e+09	0.250000	96874.625000
	I don't know	4.336084e+09	1.000000	NaN
	Other (please specify)	4.336863e+09	0.600000	55356.642857
	Roast beef	4.336174e+09	0.000000	33749.500000
	Tofurkey	4.336790e+09	0.666667	57916.166667
	Turducken	4.337475e+09	0.500000	200000.000000
	Turkey	4.336791e+09	0.531008	97690.147982
None	Chicken	4.336151e+09	0.500000	11249.500000
	Ham/Pork	4.336680e+09	0.444444	61249.500000
	I don't know	4.336412e+09	0.500000	33749.500000
	Other (please specify)	4.336688e+09	0.600000	119106.678571
	Roast beef	4.337424e+09	0.000000	162499.500000
	Tofurkey	4.336950e+09	0.500000	112499.500000
	Turducken	4.336739e+09	0.000000	NaN
	Turkey	4.336784e+09	0.523364	74606.275281
Other (please specify)	Ham/Pork	4.336465e+09	1.000000	87499.500000
	Other (please specify)	4.337335e+09	0.000000	124999.666667
	Tofurkey	4.336122e+09	1.000000	37499.500000
	Turkey	4.336724e+09	0.700000	82916.194444

As you can see above, we get a nice table that shows us the mean of each column for each group. This enables us to find some interesting patterns, such as:

```
- People who have Turducken and Homemade cranberry sauce seem to have high househol
d incomes.
- People who eat Canned cranberry sauce tend to have lower incomes, but those who a
lso have Roast Beef have the lowest incomes.
- It looks like there’s one person who has Canned cranberry sauce and doesn’t know
 what type of main dish he’s having.
```

Aggregating with multiple functions

Find sum, mean and standard deviation of each group in the income column of grouped dataframe

In [28]:

Out[28]:

What type of cranberry saucedo you typically have?	What is typically the main dish at your Thanksgiving dinner?	mean	sum	std
Canned	**Chicken**	80999.600000	404998.0	75779.481062
	Ham/Pork	77499.535714	1084993.5	56645.063944
	I don't know	4999.500000	4999.5	NaN
	Other (please specify)	53213.785714	372496.5	29780.946290
	Roast beef	25499.500000	127497.5	24584.039538
	Tofurkey	100713.857143	704997.0	61351.484439
	Turkey	85242.682045	34182315.5	55687.436102
Homemade	**Chicken**	19999.500000	59998.5	16393.596311
	Ham/Pork	96874.625000	387498.5	77308.452805
	I don't know	NaN	0.0	NaN

One of the limitations of aggregation is that each function has to return a single number. While we can perform computations like finding the mean, we can’t for example, call value_counts to get the exact count of a category. We can do this using the pandas.GroupBy.apply method. This method will apply a function to each group, then combine the results.

Find the number of people who live in each area type (Rural, Suburban, etc) who eat different kinds of main dishes for Thanksgiving

In [29]:

Out[29]:
```
How would you describe where you live?
Rural                                         Turkey                      189
                                              Other (please specify)        9
                                              Ham/Pork                      7
                                              Tofurkey                      3
                                              I don't know                  3
                                              Turducken                     2
                                              Chicken                       2
                                              Roast beef                    1
Suburban                                      Turkey                      449
                                              Ham/Pork                     17
                                              Other (please specify)       13
                                              Tofurkey                      9
                                              Roast beef                    3
                                              Chicken                       3
                                              I don't know                  1
                                              Turducken                     1
Urban                                         Turkey                      198
                                              Other (please specify)       13
                                              Tofurkey                      8
                                              Chicken                       7
                                              Roast beef                    6
                                              Ham/Pork                      4
Name: What is typically the main dish at your Thanksgiving dinner?, dtype: in
t64
```

In [ ]:

Department of Data Science - Data and Visual Analytics Lab

Lab5. Pandas Concatenate, Merge and Join

Objectives ¶

In this lab, you will learn how to

```
- concatenate two dataframes
- append a dataframe to another existing dataframe
- merge two dataframes
- join two dataframes using various SQL style join operations
```

We will play the role of a macroeconomic analyst at the Organization for Economic Cooperation and Development (OECD). The question we are trying to answer is simple but interesting: which countries have citizens putting in the longest work hours and how have these trends been changing over time?.

Unfortunately, the OECD has been collecting data for different continents and time periods separately. Our job is to first get all of the data into one place so we can run the necessary analysis.

In [1]:
```
# Import necessary modules
```

First column should be used as the row index by passing the argument index_col=0

In [2]:
```
north_america = pd.read_csv('./oecd/north_america_2000_2010.csv', index_col=0)
south_america = pd.read_csv('./oecd/south_america_2000_2010.csv', index_col=0)
```

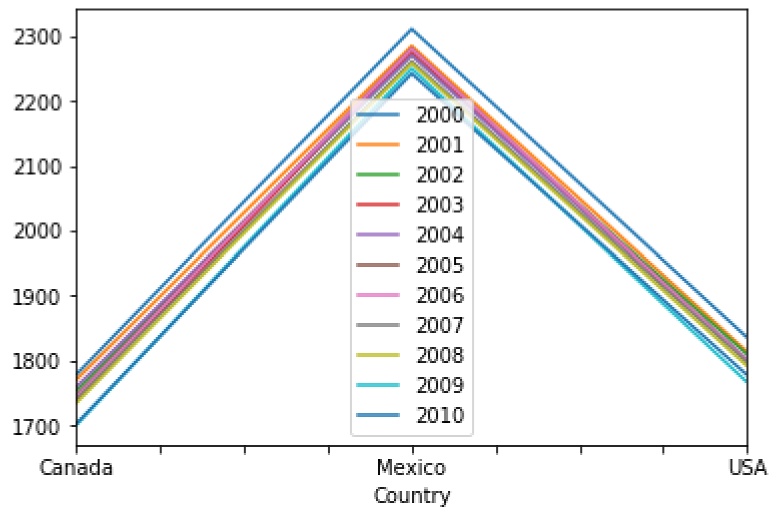
In [3]:
```
#north_america  #(UNCOMMENT AND SEE OUTPUT)
```

In [4]:
```
#south_america  #(UNCOMMENT AND SEE OUTPUT)
```

Here, rows are countries, columns are years, and cell values are the average annual hours worked per employee.

Create line graphs for our yearly labor trends in north_america

In [5]:

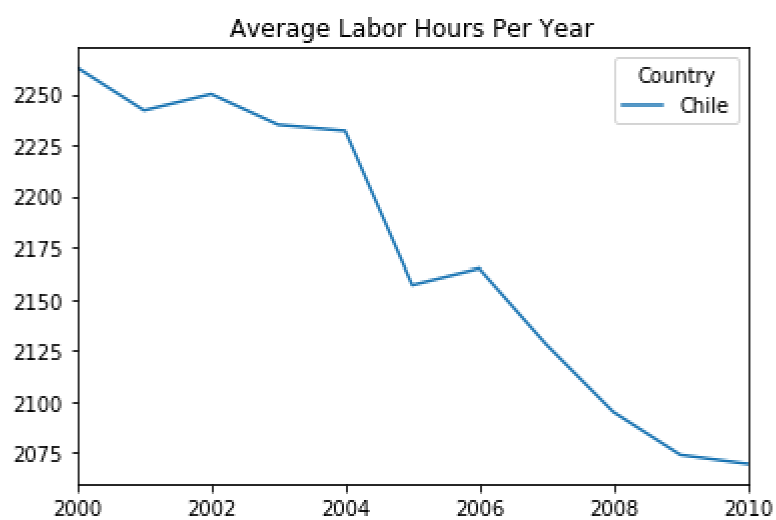


Plot transposed line graph of north_america dataframe, with title "Average Labor Hours Per Year"

In [6]:

Similarly, plot transposed south_america dataframe with title "Average Labor Hours Per Year". Output chart is shown below

In [7]:



Concatenate America Data

It's hard to compare the average labor hours in South America versus North America. If we were able to get all the countries into the same data frame, it would be much easier to do this camparison.

Concatenate north_america and south_america dataframes and store result in a dataframe, americas

In [ ]:
```
americas =
```

In [8]:
```
americas
```

Out[8]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Canada	1779.0	1771.0	1754.0	1740.0	1760.0	1747	1745.0	1741.0	1735	1701.0	1703.0
Mexico	2311.2	2285.2	2271.2	2276.5	2270.6	2281	2280.6	2261.4	2258	2250.2	2242.4
USA	1836.0	1814.0	1810.0	1800.0	1802.0	1799	1800.0	1798.0	1792	1767.0	1778.0
Chile	2263.0	2242.0	2250.0	2235.0	2232.0	2157	2165.0	2128.0	2095	2074.0	2069.6

Now, our data collection team has sent us data files for each year from 2011 to 2015 in separate CSV files. They are americas_2011.csv , americas_2012.csv, americas_2014.csv and americas_2015.csv

Load the additional files

In [9]:
```
americas_dfs = [americas]

for year in range(2011, 2016):
    filename = "./oecd/americas_{}.csv".format(year)
    df = pd.read_csv(filename, index_col=0)
    americas_dfs.append(df)
```

In [10]:
```
americas_dfs[1]
```

Out[10]:

Country	2011
Canada	1700.0
Chile	2047.4
Mexico	2250.2
USA	1786.0

In [11]:
```
#americas_dfs[2]
```

One thing you might notice is the rows in the americas_2011 DataFrame we just printed are not in the same sequence as the americas DataFrame (pandas automatically alphabetized them). Luckily, the pd.concat() function joins data on index labels (countries, in our case), not sequence, so this won't pose an issue during concatenation. If we wanted to instead concatenate the rows in the order they are currently in, we could pass the argument ignore_index=True. This would result in the indexes being assigned a sequence of integers. It's also important to keep in mind we have to create the list of DataFrames in the order we would like them concatenated, otherwise our years will be out of chronological order.

We can't use the pd.concat() function exactly the same way we did last time, because now we are adding columns instead of rows. This is where axis comes into play. By default, the argument is set to axis=0, which means we are concatenating rows. This time, we will need to pass in axis=1 to indicate we want to concatenate columns. Remember, this will only work if all the tables have the same height (number of rows).

One caveat to keep in mind when concatenating along axis 1 is the title for the row indexes, 'Country', will be dropped. This is because pandas isn't sure whether that title applies to the new row labels that have been added. We can easily fix this by assigning the DataFrame.index.names attribute.

Concatenate americas and americas_dfs dataframes and store result in americas

In [12]:
```
americas =
```

```
C:\Users\Rajkumar\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Future
Warning: Sorting because non-concatenation axis is not aligned. A future vers
ion
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """Entry point for launching an IPython kernel.
```

In [ ]:
```
americas.index.names = ['Country']
```

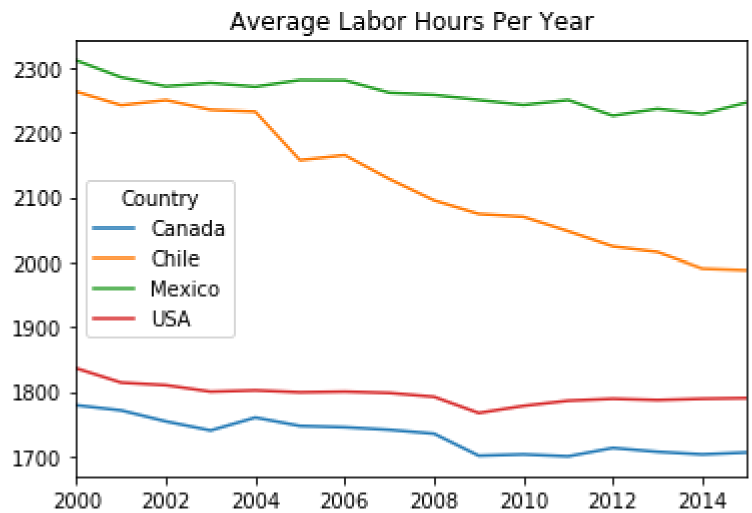
In [13]:
```
americas
```

Out[13]:

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	20
Country												
Canada	1779.0	1771.0	1754.0	1740.0	1760.0	1747	1745.0	1741.0	1735	1701.0	1703.0	170
Chile	2263.0	2242.0	2250.0	2235.0	2232.0	2157	2165.0	2128.0	2095	2074.0	2069.6	204
Mexico	2311.2	2285.2	2271.2	2276.5	2270.6	2281	2280.6	2261.4	2258	2250.2	2242.4	225
USA	1836.0	1814.0	1810.0	1800.0	1802.0	1799	1800.0	1798.0	1792	1767.0	1778.0	178

Now, plot transposed americas dataframe

In [14]:



Appending data from other Continents

The data collection team has provided CSV files for Asia, Europe, and the South Pacific for 2000 through 2015. Let's load these files in and have a preview

In [15]:

```
asia = pd.read_csv('./oecd/asia_2000_2015.csv', index_col=0)
asia
```

Out[15]:

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
Country														
Israel	2017	1979	1993	1974	1942	1931	1919	1931	1929	1927	1918	1920	1910	1867
Japan	1821	1809	1798	1799	1787	1775	1784	1785	1771	1714	1733	1728	1745	1734
Korea	2512	2499	2464	2424	2392	2351	2346	2306	2246	2232	2187	2090	2163	2079
Russia	1982	1980	1982	1993	1993	1989	1998	1999	1997	1974	1976	1979	1982	1980

In [16]:
```python
europe = pd.read_csv('./oecd/europe_2000_2015.csv', index_col=0)
europe.head()
```

Out[16]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	201
Austria	1807.4	1794.6	1792.2	1783.8	1786.8	1764.0	1746.2	1736.0	1728.5	1673.0	1668.
Belgium	1595.0	1588.0	1583.0	1578.0	1573.0	1565.0	1572.0	1577.0	1570.0	1548.0	1546.
Switzerland	1673.6	1635.0	1614.0	1626.8	1656.5	1651.7	1643.2	1632.7	1623.1	1614.9	1612.
Czech Republic	1896.0	1818.0	1816.0	1806.0	1817.0	1817.0	1799.0	1784.0	1790.0	1779.0	1800.
Germany	1452.0	1441.9	1430.9	1424.8	1422.2	1411.3	1424.7	1424.4	1418.4	1372.7	1389.

In [17]:
```python
south_pacific = pd.read_csv('./oecd/south_pacific_2000_2015.csv', index_col=0)
south_pacific
```

Out[17]:

Country	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	
Australia	1778.7	1736.7	1731.7	1735.8	1734.5	1729.2	1720.5	1712.5	1717.2	1690	1691.5	1
New Zealand	1836.0	1825.0	1826.0	1823.0	1830.0	1815.0	1795.0	1774.0	1761.0	1740	1755.0	1

If any columns were missing from the data we are trying to append, they would result in those rows having NaN values in the cells falling under the missing year columns. Let's run the append method and verify that all the countries have been sucesfully appended by printing DataFrame.index.

Append asia, europe and south_pacific to americas dataframe and assign to new dataframe world

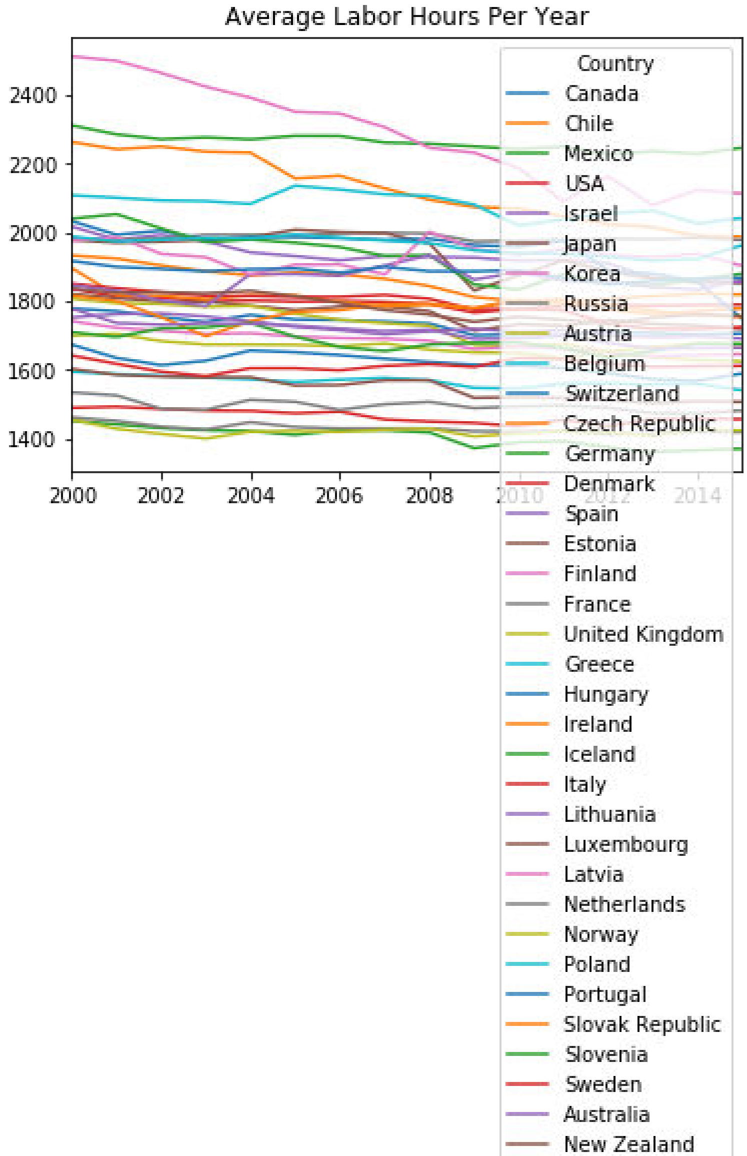
In [18]:
```python
world =
```

In [19]:
```python
world.index
```

Out[19]:
```
Index(['Canada', 'Chile', 'Mexico', 'USA', 'Israel', 'Japan', 'Korea',
       'Russia', 'Austria', 'Belgium', 'Switzerland', 'Czech Republic',
       'Germany', 'Denmark', 'Spain', 'Estonia', 'Finland', 'France',
       'United Kingdom', 'Greece', 'Hungary', 'Ireland', 'Iceland', 'Italy',
       'Lithuania', 'Luxembourg', 'Latvia', 'Netherlands', 'Norway', 'Polan
d',
       'Portugal', 'Slovak Republic', 'Slovenia', 'Sweden', 'Australia',
       'New Zealand'],
      dtype='object', name='Country')
```

Plot, transposed world dataframe

```
In [20]:
```

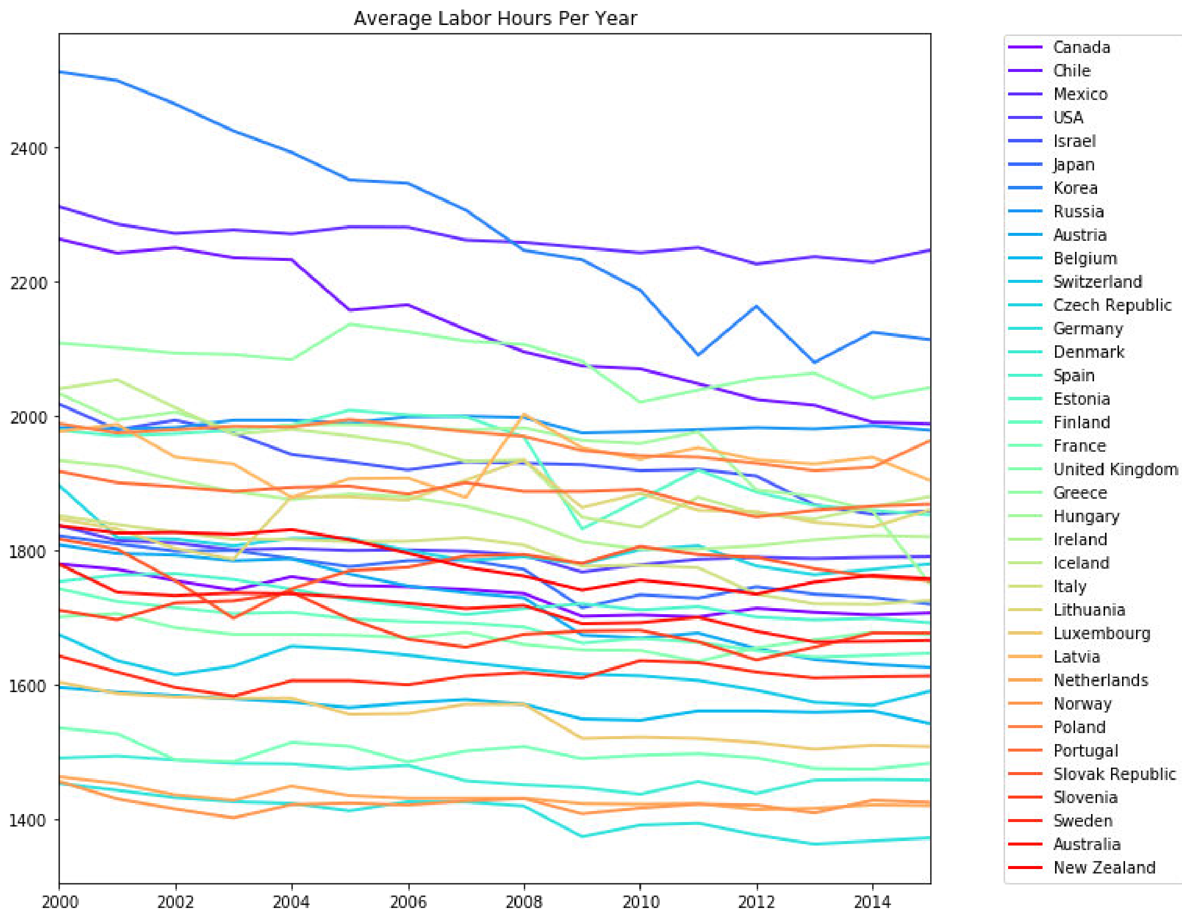


let us customize this plot, so that country names appear outside the chart

Update plot() with the following features

```
figsize=(10,10),
colormap='rainbow',
linewidth=2,
loc='right'
```

In [21]:



Merging Historical Labor Data

It's nice being able to see how the labor hours have shifted since 2000, but in order to see real trends emerge, we want to be able to see as much historical data as possible. The data collection team was kind enough to send data from 1950 to 2000, let's load it in and take a look.

In [22]:
```
historical = pd.read_csv('./oecd/historical.csv', index_col=0)
historical.head()
```

Out[22]:

Country	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	...	1990	1991
Australia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1779.5	1774.90
Austria	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
Belgium	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1662.9	1625.79
Canada	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1789.5	1767.50
Switzerland	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	1673.10

5 rows × 50 columns

You'll notice there are a lot of NaN values, especially in the earlier years. This simply means that there was no data collected for those countries in the earlier years. Putting a 0 in those cells would be misleading, as it would imply that no one spent any hours working that year! Instead, NaN represents a null value, meaning "not a number". Having null values will not affect our DataFrame merging since we will use the row labels (index) as our key.

When merging, it's important to keep in mind which rows will be retained from each table. I'm not sure what the full dimensions of my tables are, so instead of displaying the whole thing, we can just look at facts we're interested in. Let's print the DataFrame.shape() attribute to see a tuple containing (total rows, total columns) for both tables.

In [23]:
```
print("World rows & columns: ", world.shape)
print("Historical rows & columns: ", historical.shape)
```

```
World rows & columns:  (36, 16)
Historical rows & columns:  (39, 50)
```

Note that the historical table has 39 rows, even though we are only analyzing 36 countries in our world table. Dropping the three extra rows can be automatically taken care of with some proper DataFrame merging. We will treat world as our primary table and want this to be on the right side of the resulting DataFrame and historical on the left, so the years (columns) stay in chronological order. The columns in these two tables are all distinct, that means we will have to find a key to join on. In this case, the key will be the row indexes (countries).

We will want to do a right join using the pd.merge() function and use the indexes as keys to join on.

The right join will ensure we only keep the 36 rows from the right table and discard the extra 3 from the historical table. Let's print the shape of the resulting DataFrame and display the head to make sure everything turned out correct.

Merge historical dataframe with world dataframe and store in a new variable, world_historical

In [25]:
```
world_historical =
```

Print size of world_historical dataframe

In [26]:
```
```

```
(36, 66)
```

Print top-5 of world_historical dataframe

In [27]:
```
```

Out[27]:

	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	...	200
Country												
Canada	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1745
Chile	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2165
Mexico	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2280
USA	1960.0	1975.5	1978.0	1980.0	1970.5	1992.5	1990.0	1962.0	1936.5	1947.0	...	1800
Israel	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1919

5 rows × 66 columns

Joining Historical Data

Now that we've done it the hard way and understand table merging conceptually, let's try a more elegant technique. Pandas has a clean method to join on indexes which is perfect for our situation.

Use join method to join historical dataframe and world dataframe and store result in world_historical dataframe

In [28]:
```
world_historical =
```

In [29]:
```
# Print head of world_historical dataframe
```

Out[29]:

	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	...	200
Country												
Canada	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1745
Chile	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2165
Mexico	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2280
USA	1960.0	1975.5	1978.0	1980.0	1970.5	1992.5	1990.0	1962.0	1936.5	1947.0	...	1800
Israel	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	1919

5 rows × 66 columns

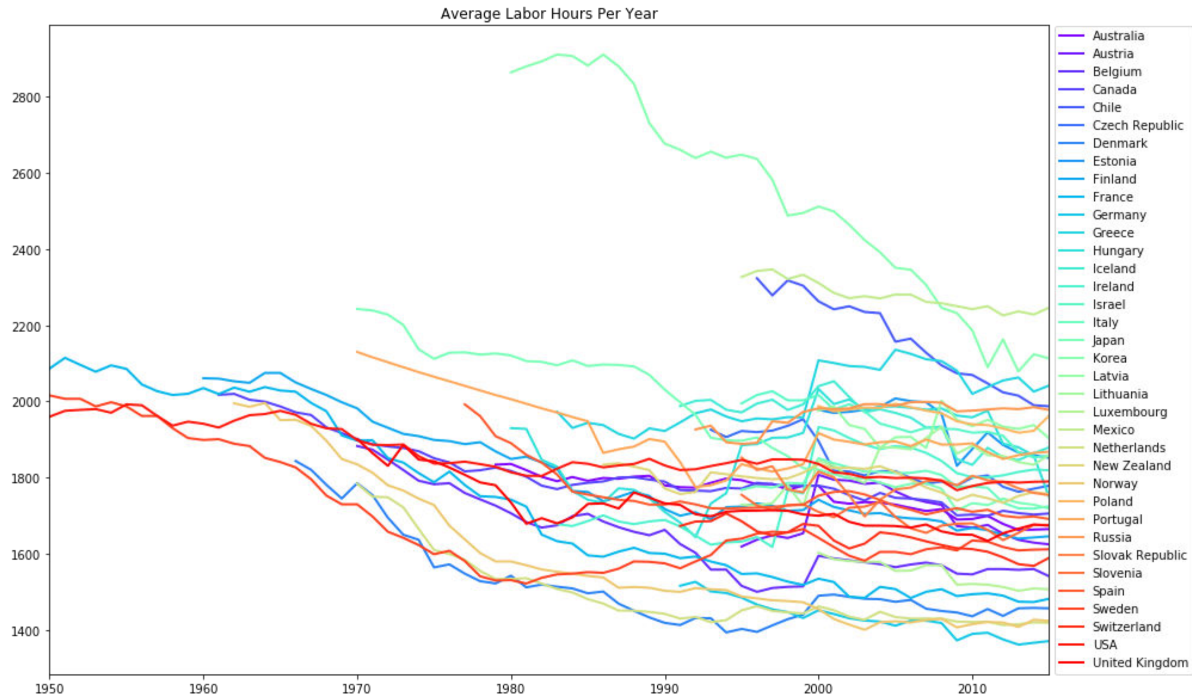
Plot our world labor data

Before plotting the final line graph, it's a good idea to sort our rows alphabetically to make the legend more easy to read for our viewers. This can be executed with the DataFrame.sort_index() method. We can pass in the parameter inplace=True to avoid having to reassign our world_historical variable.

In [30]:
```

```

Plot, transposed world_historical dataframe

In [31]:



Which country worked longer hours per year?

In [ ]:

Which country worked shorter hours per year?

In [ ]:

Department of Data Science - Data and Visual Analytics Lab

Lab6. Pandas Data Cleaning

Objectives

In this lab, you will learn how to

- Clean Column Names
- Converte String Columns to Numeric
- Remove Non-Digit Characters
- Convert Columns to Numeric Dtypes
- Rename Columns
- Extract Values from Strings
- Drop Missing Values
- Fill Missing Values

Data Cleaning Steps

Verify the contents with .head() method

```
import pandas as pd
df = pd.read_csv('path_to_data')
df.head(10)
```

See the names and types of the columns. Most of the time you're going get data that is not quite what you expected, such as dates which are actually strings

```
#Get column names
column_names = df.columns
print(column_names)

#Get column data types
df.dtypes

#Also check if the column is unique
for i in column_names:
   print('{} is unique: {}'.format(i, df[i].is_unique))
```

Now let's see if the dataframe has an index associated with it, by calling .index on the df.

```
# Check the index values
df.index.values

# Check if a certain index exists
'foo' in df.index.values

# If index does not exist
df.set_index('column_name_to_use', inplace=True)
```

Now, let's figure out which columns you want to keep or remove. We want to remove the columns in indexes 1, 3, and 5

```
# Create list comprehension of the columns you want to lose
columns_to_drop = [column_names[i] for i in [1, 3, 5]]
```

```
#Drop unwanted columns
df.drop(columns_to_drop, inplace=True, axis=1)
```

The **inplace=True** has been added so you don't need to save over the original df by assigning the result of .drop() to df.

What To Do With NaN

If you need to fill in errors or blanks, use the fillna() and dropna() methods. It seems quick, but all manipulations of the data should be documented so you can explain them to someone at a later time.

You could fill the NaNs with strings, or if they are numbers you could use the mean or the median value. There is a lot of debate on what do with missing or malformed data, and the correct answer is … it depends.

You'll have to use your best judgement and input from the people you're working with on why removing or filling the data is the best approach.

```
# Fill NaN with ' '
df['col'] = df['col'].fillna(' ')

#Fill NaN with 99
df['col'] = df['col'].fillna(99)

#Fill NaN with the mean of the column
df['col'] = df['col'].fillna(df['col'].mean())
```

You can also propagate non-null values forward or backwards by putting method='pad' as the method argument. It will fill the next value in the dataframe with the previous non-NaN value. Maybe you just want to fill one value (limit=1)or you want to fill all the values. Whatever it is make sure it is consistent with the rest of your data cleaning

```
df = pd.DataFrame(data={'col1':[np.nan, np.nan, 2,3,4, np.nan,
np.nan]})

     col1
0   NaN
1   NaN
2   2.0
3   3.0
4   4.0   # This is the value to fill forward
5   NaN
6   NaN

df.fillna(method='pad', limit=1)

     col1
0   NaN
1   NaN
2   2.0
3   3.0
4   4.0
5   4.0 # Filled forward
6   NaN
```

Notice how only index 5 was filled? If I had not filled limited the pad , it would have filled the entire dataframe. We are not limited to forward filling, but also backfilling with bfill.

Fill the first two NaN values with the first available value

```
df.fillna(method='bfill')

     col1
0   2.0 # Filled
1   2.0 # Filled
2   2.0
3   3.0
4   4.0
5   NaN
6   NaN
```

You could just drop them from the dataframe entirely, either by the row or by the column.

```
# Drop any rows which have any nans
df.dropna()

# Drop columns that have any nans
df.dropna(axis=1)

# Only drop columns which have at least 90% non-NaNs
df.dropna(thresh=int(df.shape[0] * .9), axis=1)
```

np.where

Consider if you're evaluating a column, and you want to know if the values are strictly greater than 10. If they are you want the result to be 'foo' and if not you want the result to be 'bar'

```
# Follow this syntax
  np.where(if_this_condition_is_true, do_this, else_this)

  # Example
  df['new_column'] = np.where(df[i] > 10, 'foo', 'bar)
```

You're able to do more complex operations like the one below. Here we are checking if the column record starts with foo and does not end with bar. If this checks out we will return True else we'll return the current value in the column.

```
df['new_column'] = np.where(df['col'].str.startswith('foo') and
                            not df['col'].str.endswith('bar'),
                            True, df['col'])
```

And even more effective, you can start to nest your np.where so they stack on each other. Similar to how you would stack ternary operations, make sure they are readable as you can get into a mess quickly with heavily nested statements.

```
# Three level nesting with np.where
  np.where(if_this_condition_is_true_one, do_this,
    np.where(if_this_condition_is_true_two, do_that,
      np.where(if_this_condition_is_true_three, do_foo, do_bar)))

#A trivial example
df['foo'] = np.where(df['bar'] == 0, 'Zero',
               np.where(df['bar'] == 1, 'One',
                    np.where(df['bar'] == 2, 'Two', 'Three')))
```

Assert and Test What You Have

Just because you have your data in a nice dataframe, no duplicates, no missing values, you still might have some issues with the underlying data. And, with a dataframe of 10M+ rows or new API, how can you make sure the values are exactly what you expect them to be?

Truth is, you never really know if your data is correct until you test it. Best practices in software engineering rely heavily on testing their work, but for data science it is still a work in progress. Better to start now and teach yourself good work principles, rather than having to retrain yourself at a later date.

Let's make a simple dataframe to test.

```
df = pd.DataFrame(data={'col1':np.random.randint(0, 10, 10),
      'col2':np.random.randint(-10, 10, 10)})
>>
   col1  col2
0     0     6
1     6    -1
2     8     4
3     0     5
4     3    -7
5     4    -5
6     3   -10
7     9    -8
8     0     4
9     7    -4
```

Let's test if all the values in col1 are >= 0 by using the built in method assert which comes with the standard library in python. What you're asking python if is True all the items in df['col1'] are greater than zero. If this is True then continue on your way, if not throw an error

```
assert(df['col1'] >= 0 ).all() # Should return nothing
```

Humm looks like we have some options when we're testing our dataframes. Let's test if any of the values are strings.

```
assert(df['col1'] != str).any() # Should return nothing
```

The best practice with asserts is to be used to test conditions within your data that should never happen. This is so when you're running your code, everything stops should one of these assertions fail.

The .all() method will check if all the elements in the objects pass the assert, while .any() will check if any of the elements in the objects pass the assert test.

This can be helpful when you want to:

- Check if any negative values have been introduced into the data;
- Make sure two columns are exactly the same;
- Determine the results of a transformation, or;
- Check if unique id count is accurate.

Pandas Testing Package

Not only do we get an error thrown, but pandas will tell us what is wrong

```
import pandas.util.testing as tm
tm.assert_series_equal(df['col1'], df['col2'])
>>
AssertionError: Series are different. Series values are different
(100.0 %)
[left]:  [0, 6, 8, 0, 3, 4, 3, 9, 0, 7]
[right]: [6, -1, 4, 5, -7, -5, -10, -8, 4, -4]
```

Additionally, if you want to start building yourself a testing suite — and you might want to think about doing this — get familiar with the unittest package built into the Python library

beautifier

Instead of having to write your own regex — which is a pain at the best of times — sometimes it's been done for you. The beautifier package is able to help you clean up some commonly used patterns for emails or URLs. It's nothing fancy but can quickly help with clean up.

Install beautifier first using: pip3 install beautifier

```
from beautifier import Email, Url
email_string = 'foo@bar.com'
email = Email(email_string)
print(email.domain)
print(email.username)
print(email.is_free_email)

Output:
bar.com
foo
False
url_string =
'https://github.com/labtocat/beautifier/blob/master/beautifier/__init__.py
'
url = Url(url_string)
print(url.param)
print(url.username)
print(url.domain)

Output:
None
{'msg': 'feature is currently available only with linkedin urls'}
  github.com
```

Dealing with Unicode

When doing some NLP, dealing with Unicode can be frustrating at the best of times. I'll be running something in spaCy and suddenly everything will break on me because of some unicode character appearing somewhere in the document body.

It really is the worst.

By using using ftfy (fixed that for you) you're able to fix really broken Unicode. Consider when someone has encoded Unicode with one standard and decoded it with a different one. Now you have to deal with this in between string, as nonsense sequences called "mojibake".

```
# Example of mojibake
  ¯\\_(ã\x83\x84)_/¯
  \ufeffParty
  \001\033[36;44mI’m
```

Let’s see what our strings above can be converted into, so we can read it. The main method is fix_text(), and you’ll use that to perform the decoding.

```
import ftfy

foo = '¯\\_(ã\x83\x84)_/¯'
bar = '\ufeffParty'
baz = '\001\033[36;44mI’m'

print(ftfy.fix_text(foo))
print(ftfy.fix_text(bar))
print(ftfy.fix_text(baz))
```

Department of Data Science - Data and Visual Analytics Lab

Lab6. Pandas Data Cleaning Part-II

LabelEncoder in Scikit Learn

Encodes string values as integer values

In [1]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
le = LabelEncoder()

#New object
df = pd.DataFrame(data = {'col1': ['foo','bar','foo','bar'],
                          'col2': ['x', 'y', 'x', 'z'],
                          'col3': [1, 2, 3, 4]})
```

In [3]:
```
#Now convert string values of each column into integer values
df.apply(le.fit_transform)
```

Out[3]:

	col1	col2	col3
0	1	0	0
1	0	1	1
2	1	0	2
3	0	2	3

One Hot Encoder

Consider the following dataframe. You will have to represent string values of column A and B with integers

In [4]:
```python
import pandas as pd
df = pd.DataFrame({'A': ['a', 'b', 'a'], 'B': ['b', 'a', 'c'], 'C': [1, 2, 3
]})
df
```

Out[4]:

	A	B	C
0	a	b	1
1	b	a	2
2	a	c	3

In [5]:
```python
# Call get_dummies method. It will create a new column for each string value i
n DF columns
pd.get_dummies(df, prefix=['col1', 'col2']) # here prefix tells which columns
 shoduld be encoded
```

Out[5]:

	C	col1_a	col1_b	col2_a	col2_b	col2_c
0	1	1	0	0	1	0
1	2	0	1	1	0	0
2	3	1	0	0	0	1

MinMaxScaler

It will transform values into a range of 0 to 1

In [6]:
```python
from sklearn.preprocessing import MinMaxScaler
mm_scaler = MinMaxScaler(feature_range=(0, 1))  # (0,1) is default range

df2 = pd.DataFrame({"col1":[5, -41, -67],
                    "col2":[23, -53, -36],
                    "col3":[-25,  10,  17] })

mm_scaler.fit_transform(df2)
```

```
C:\Users\Rajkumar\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:3
34: DataConversionWarning: Data with input dtype int64 were all converted to
float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

Out[6]:
```
array([[1.        , 1.        , 0.        ],
       [0.36111111, 0.        , 0.83333333],
       [0.        , 0.22368421, 1.        ]])
```

Binarizer

It will encode values into 0 or 1, depending on the threshold

In [7]:
```
from sklearn.preprocessing import Binarizer
dfb = pd.DataFrame({ "col1": [110, 200],
                     "col2": [120, 800],
                     "col3": [310, 400] })

bin = Binarizer(threshold=300)
bin.fit_transform(dfb)
```

Out[7]:
```
array([[0, 0, 1],
       [0, 1, 1]], dtype=int64)
```

Imputer

You can also use Imputer from sklearn to handle NaN objects in each columns. Here, we replace NaN with column mean value. This is good alternative to fillna() method.

In [8]:
```
import numpy as np
from sklearn.impute import SimpleImputer
import pandas as pd

imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')

df = pd.DataFrame( {"col1": [7, 2, 3],
                    "col2": [4, np.nan, 6],
                    "col3": [np.nan, np.nan, 3],
                    "col4": [10, np.nan, 9] })
print(df)

imp_mean.fit_transform(df)
```

```
   col1  col2  col3  col4
0     7   4.0   NaN  10.0
1     2   NaN   NaN   NaN
2     3   6.0   3.0   9.0
```

Out[8]:
```
array([[ 7. ,  4. ,  3. , 10. ],
       [ 2. ,  5. ,  3. ,  9.5],
       [ 3. ,  6. ,  3. ,  9. ]])
```

De-duplication or Entity Resolution and String Matching

You can use **dedupe** and **fuzzywuzzy** packages. Install them using pip3 and import inside your Python code

Conclusion: Life is not just a bunch of Kaggle datasets, where in reality you'll have to make decisions on how to access and clean the data you need everyday. Sometimes you'll have a lot of time to make sure everything is in the right place, but most of the time you'll be pressed for answers. If you have the right tools in place and understanding of what is possible, you'll be able to get to those answers easily.

```
In [ ]:
```

Department of Data Science - Data and Visual Analytics Lab

Lab7. Data Visualization in Seaborn

Objectives

After completing this lab, you will learn how to

```
- Visualize Statistical Relationships using Scatter plot, relplot, Hue plot, Line p
lot
- Plot Categorical Data using Jitter plot, swarm plot, violin plot, box plot, point
plot
- Visualize the distribution of dataset using Histogram, Hexplot, KDE plot, Boxen p
lot
- perform Pairwise correlation using Heatmap
- Understand Multiple bivariate relationships using Pairplot
```

Dataset - Online Question and Answer Platform

An online question and answer platform has hired you as a data scientist to identify the best question authors on the platform. This identification will bring more insight into increasing the user engagement. The tag of the question, number of views received, number of answers, username and reputation of the question author are given in this dataset. The problem requires you to predict the upvote count that the question will receive.

```
Variable    Definition
ID          Question ID
Tag         Anonymised tags representing question category
Reputation  Reputation score of question author
Answers     Number of times question has been answered
Username    Anonymised user id of question author
Views       Number of times question has been viewed
Upvotes     (Target) Number of upvotes for the question
```

In [1]:
```
# Import necessary packages
```

1. Visualizing Statistical Relationships

A statistical relationship denotes a process of understanding relationships between different variables in a dataset and how that relationship affects or depends on other variables.

Here, we'll be using seaborn to generate the below plots:

```
Scatter plot
relplot
Hue plot
Line plot
```

In this exercise, let us Predict the number of upvotes

Import train_upvote_mini.csv file

In [2]:
```
df = pd.read_csv("./visualization_data/train_upvote_mini.csv")
df.head()
```

Out[2]:

	ID	Tag	Reputation	Answers	Username	Views	Upvotes
0	52664	a	3942.0	2.0	155623	7855.0	42.0
1	327662	a	26046.0	12.0	21781	55801.0	1175.0
2	468453	c	1358.0	4.0	56177	8067.0	60.0
3	96996	a	264.0	3.0	168793	27064.0	9.0
4	131465	c	4271.0	4.0	112223	13986.0	83.0

What is its size?

In [13]:
```

```

Out[13]: (15440, 7)

Show the types of each feature

In [ ]:
```

```

How many unique "tag" available?

In [ ]:

Visualize with Scatterplot

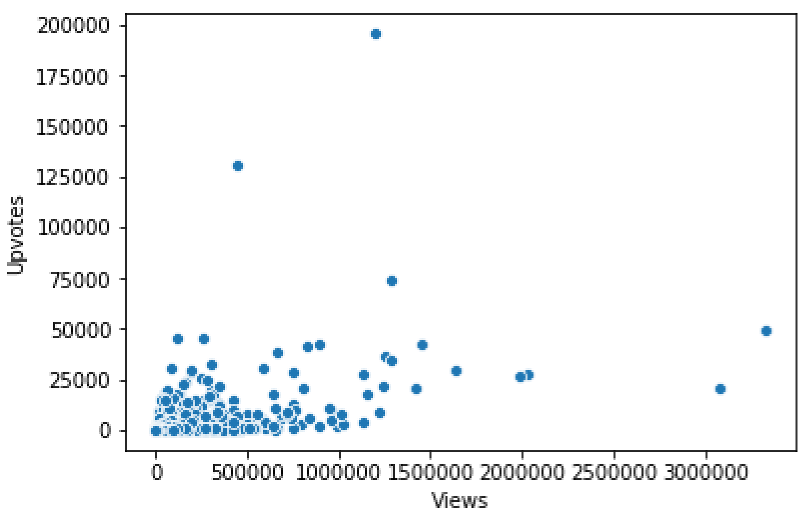
A scatterplot is perhaps the most common example of visualizing relationships between two variables. Each point shows an observation in the dataset and these observations are represented by dot-like structures. The plot shows the joint distribution of two variables using a cloud of points.

Does no. of views correlate no of upvotes?.

Show scatterplot (inherited from matplotlib) and relplot between "views" and "upvotes"

In [3]:

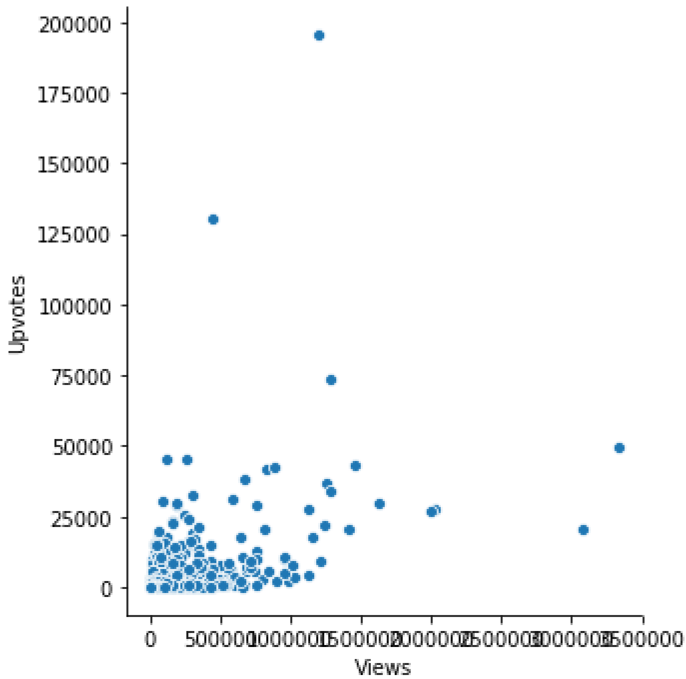
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x1c9f75f00b8>



Plot replot between "Views" and "Upvotes"

In [4]:

Out[4]: <seaborn.axisgrid.FacetGrid at 0x1c9f9681518>

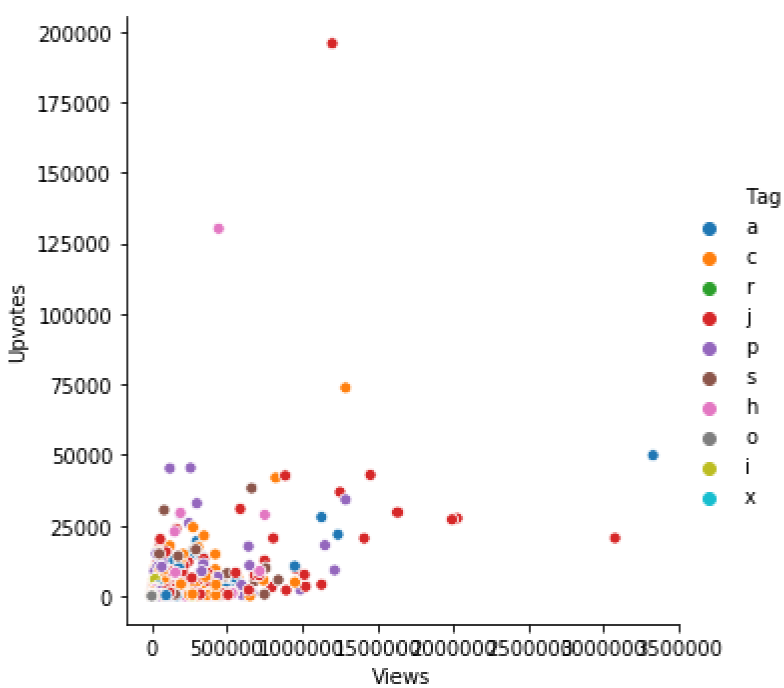


Next, we want to see the tag associated with data.

Plot relplot between "Views" and "Upvotes" with hue as "Tag"

In [5]:

Out[5]: <seaborn.axisgrid.FacetGrid at 0x1c9f9725898>



Hue Plot

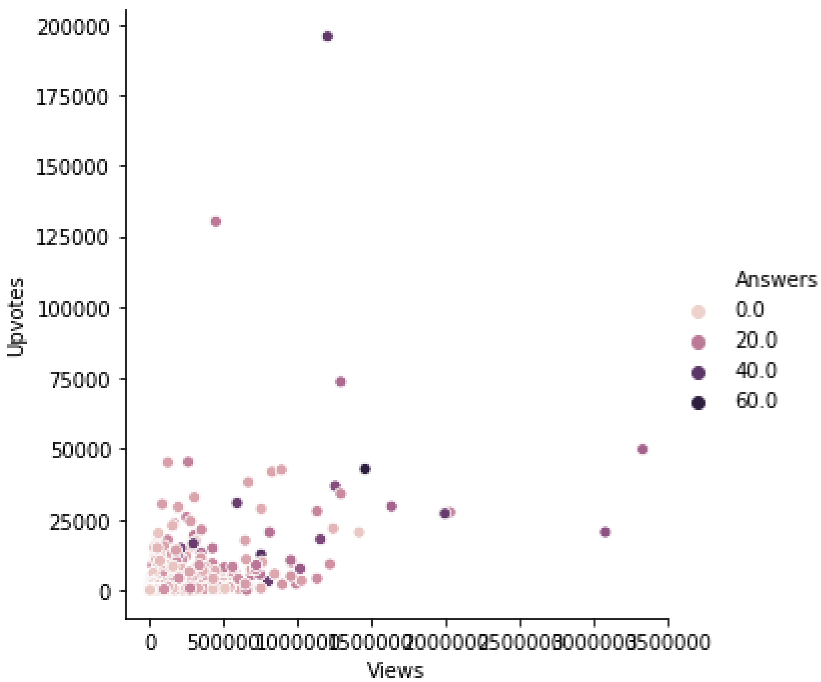
We can add another dimension in our plot with the help of hue as it gives color to the points and each color has some meaning attached to it.

In the above plot, the hue semantic is categorical. That's why it has a different color palette. If the hue semantic is numeric, then the coloring becomes sequential.

Plot relplot between "Views" and "Upvotes" with hue as "Answers"

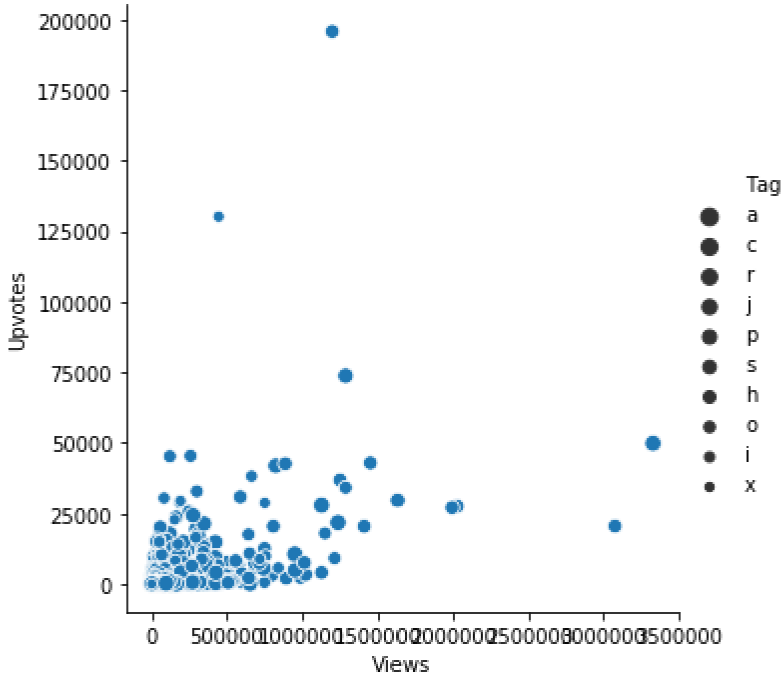
In [6]:

```
Out[6]: <seaborn.axisgrid.FacetGrid at 0x1c9f9772b70>
```



Plot relplot between "Views" and "Upvotes" with size as "Tag"

In [7]:

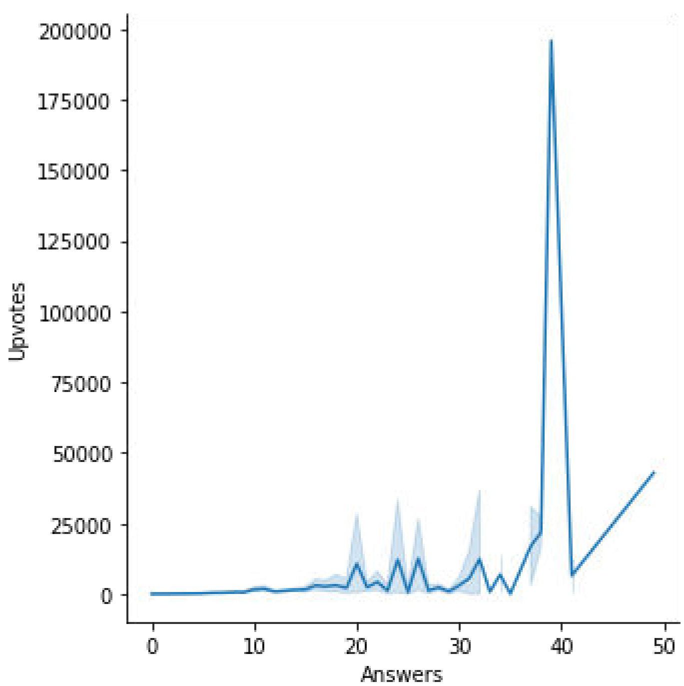


Does no of times question answered impact the no. of upvotes?

Plot line chart using relplot between "Answers" and "Upvotes"

In [8]:

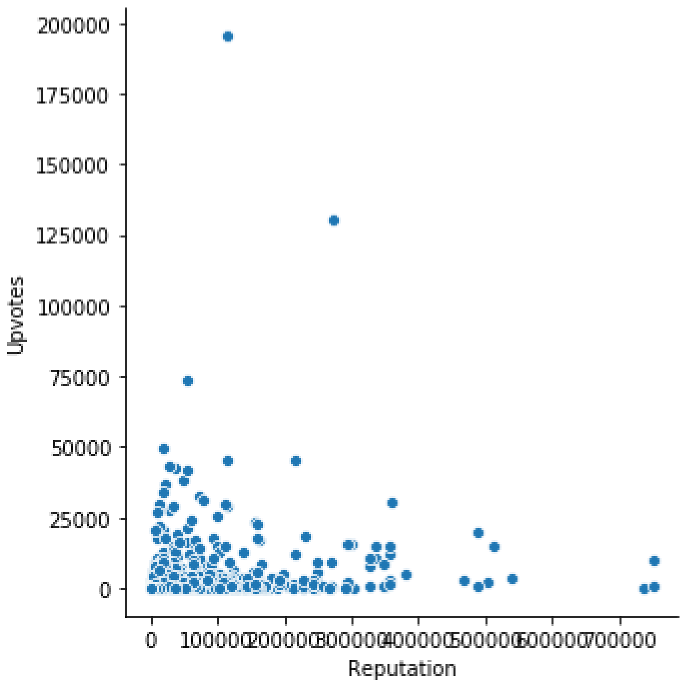
Out[8]: <seaborn.axisgrid.FacetGrid at 0x1c9f99a0c88>



Does Reputation score of question author impact no of upvotes?. Draw replot.

In [9]:

Out[9]: <seaborn.axisgrid.FacetGrid at 0x1c9f9b03898>



2. Visualizating Categorial Data

Various Categorial Plots in Seaborn

Categorical scatterplots:

```
- stripplot() (with kind="strip"; the default)
- swarmplot() (with kind="swarm")
```

Categorical distribution plots:

```
- boxplot() (with kind="box")
- violinplot() (with kind="violin")
- boxenplot() (with kind="boxen")
```

Categorical estimate plots:

```
- pointplot() (with kind="point")
- barplot() (with kind="bar")
- countplot() (with kind="count")
```

In the previous section, we saw how we can use different visual representations to show the relationship between multiple variables. We drew the plots between two numeric variables. In this section, we'll see the relationship between two variables of which one would be categorical (divided into different groups).

We'll be using catplot() function of seaborn library to draw the plots of categorical data using HR Analytics Dataset.

Dataset - HR analytics description

Your client is a large MNC and they have 9 broad verticals across the organisation. One of the problem your client is facing is around identifying the right people for promotion (only for manager position and below) and prepare them in time. Currently the process, they are following is:

1. They first identify a set of employees based on recommendations/ past performance
2. Selected employees go through the separate training and evaluation program for each vertical. These programs are based on the required skill of each vertical
3. At the end of the program, based on various factors such as training performance, KPI completion (only employees with KPIs completed greater than 60% are considered) etc., employee gets promotion

For above mentioned process, the final promotions are only announced after the evaluation and this leads to delay in transition to their new roles. Hence, company needs your help in identifying the eligible candidates at a particular checkpoint (ie., time frame from the time of nomination stage to a particular time point) so that they can expedite the entire promotion cycle.

They have provided multiple attributes around Employee's past and current performance along with demographics. Now, The task is to predict whether a potential promotee at checkpoint in the test set will be promoted or not after the evaluation process.

```
Features of HR analytics dataset
Variable    Definition
employee_id Unique ID for employee
department  Department of employee
region  Region of employment (unordered)
education   Education Level
gender  Gender of Employee
recruitment_channel Channel of recruitment for employee
no_of_trainings no of other trainings completed in previous year on soft skills, technical
skills etc.
age Age of Employee
previous_year_rating    Employee Rating for the previous year
length_of_service   Length of service in years
KPIs_met >80%   if Percent of KPIs(Key performance Indicators) >80% then 1 else 0
awards_won? if awards won during previous year then 1 else 0
avg_training_score  Average score in current training evaluations
is_promoted (Target) Recommended for promotion
```

Jitter Plot

For jitter plot we'll be using another dataset from the problem HR analysis challenge, let's import the dataset now.

In [11]:
```
df2 = pd.read_csv("./visualization_data/train_hr_mini.csv")
df2.head()
```

Out[11]:

	employee_id	department	region	education	gender	recruitment_channel	no_of_trainings
0	65438	Sales & Marketing	region_7	Master's & above	f	sourcing	1
1	65141	Operations	region_22	Bachelor's	m	other	1
2	7513	Sales & Marketing	region_19	Bachelor's	m	sourcing	1
3	2542	Sales & Marketing	region_23	Bachelor's	m	other	2
4	48945	Technology	region_26	Bachelor's	m	other	1

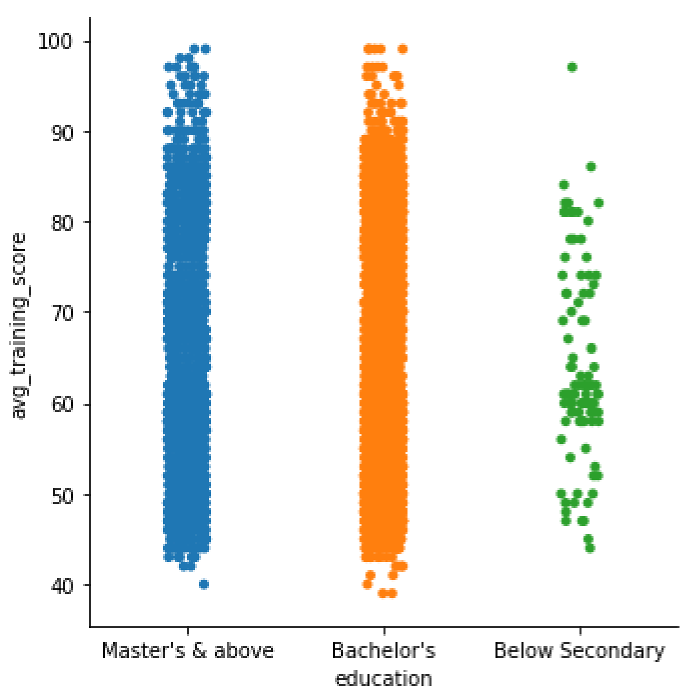
In [12]:

Out[12]: (6397, 14)

Show Jitter plot between education and avg_training_score

In [14]:

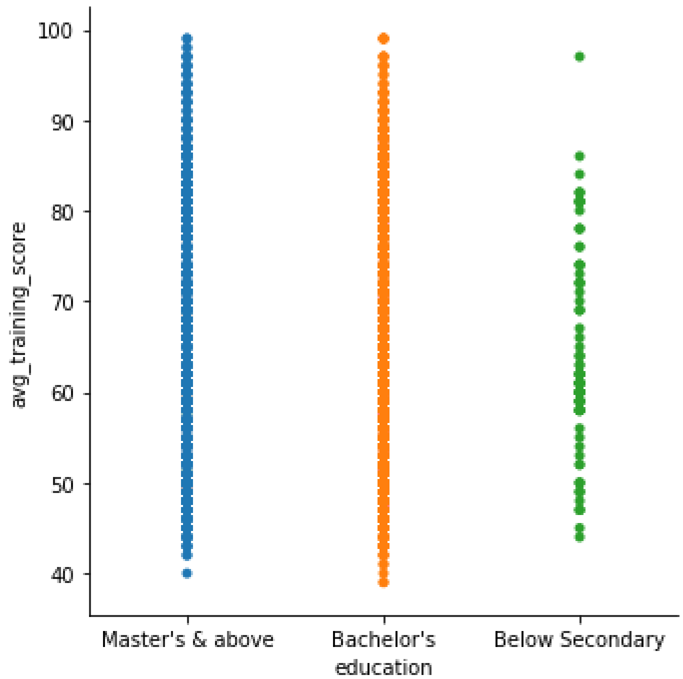
Out[14]: <seaborn.axisgrid.FacetGrid at 0x1c9fadc67f0>



Here, there are a lot of deviation from true values of the points that is called Jitter. So, let us make Jitter to false and visualize data.

In [15]:

Out[15]: <seaborn.axisgrid.FacetGrid at 0x1c9fae66cf8>



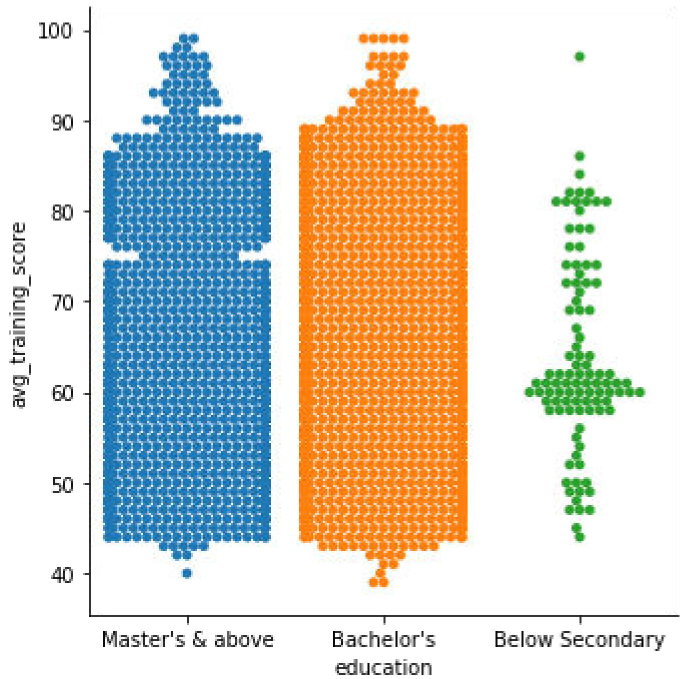
Sworm Plot

Swarm plot adjusts the points along the categorical axis using an algorithm that prevents them from overlapping. It can give a better representation of the distribution of observations.

Plot Swarm plot between education category and avg_training_score

In [16]:

Out[16]: <seaborn.axisgrid.FacetGrid at 0x1c9fae66ba8>



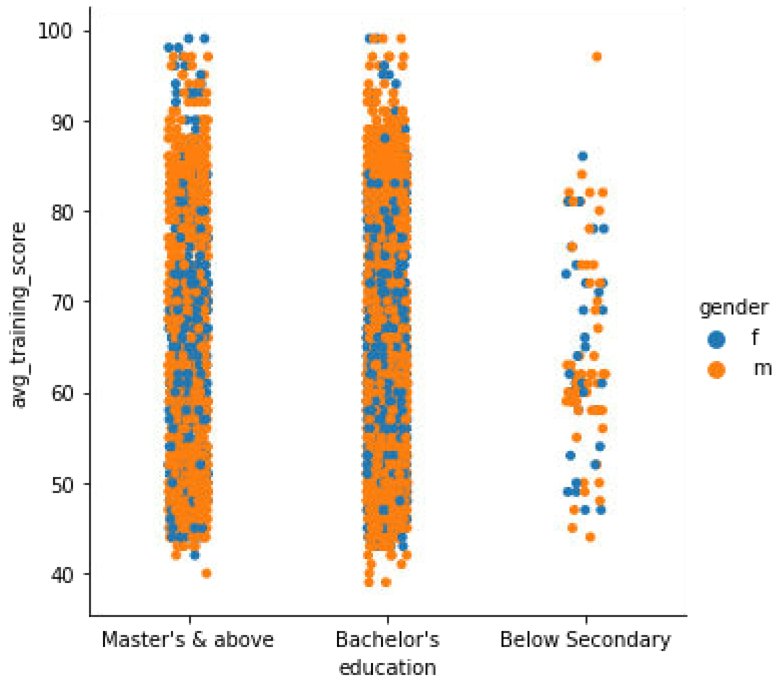
Hue Plot

Now we want to introduce another variable or another dimension in our plot, we can use the hue parameter. We want to see the gender distribution in the plot of education category and avg_training_score

Show Hue Plot to see the gender distribution in the plot of education category and avg_training_score. Here, hue is "gender".

In [17]:

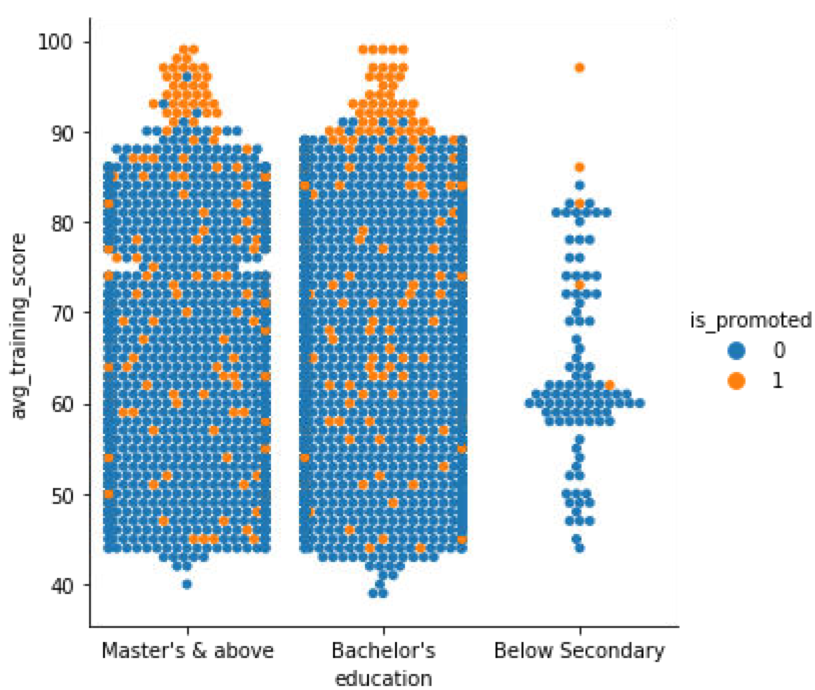
Out[17]: <seaborn.axisgrid.FacetGrid at 0x1c9faed7e80>



Who are all promoted considering education and avg training score?. Draw swarm plot with hue as "is_promoted"

In [18]:

Out[18]: <seaborn.axisgrid.FacetGrid at 0x1c9faebef28>



From this plot, we can clearly see people with higher scores got a promotion.

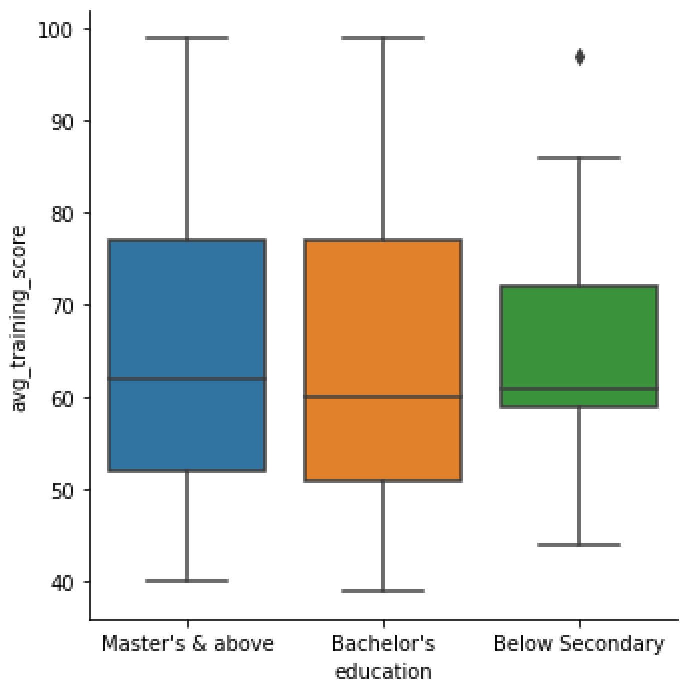
Box Plot

Boxplot shows three quartile values of the distribution along with the end values. Each value in the boxplot corresponds to actual observation in the data.

Draw box plot between education and avg_training_score

In [20]:

Out[20]: <seaborn.axisgrid.FacetGrid at 0x1c9f998c208>



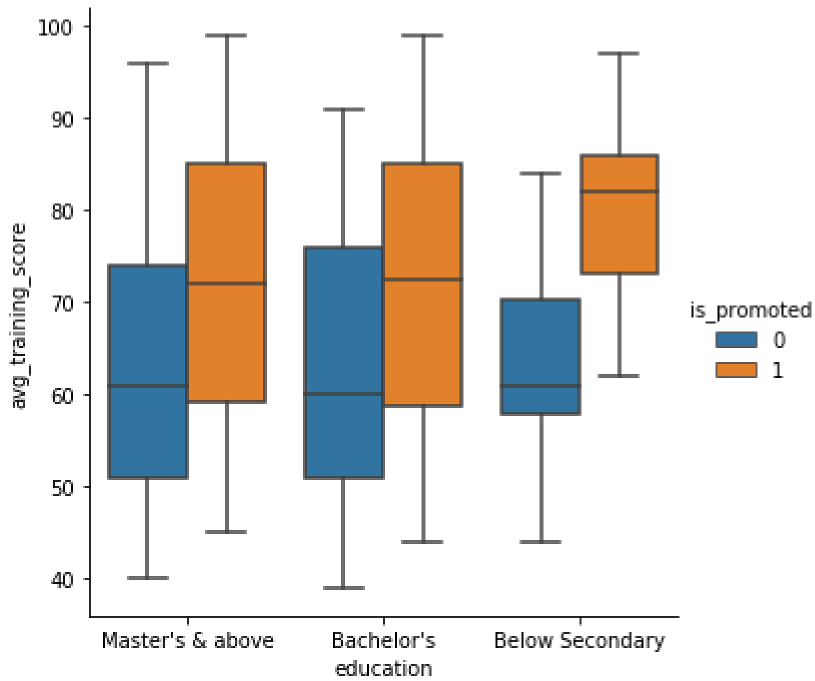
From this chart, we can understand that promotees with masters degree have a minimum of 40, maximum of 100 scores and average score of around 62. Similarly, we can see the 25th and 75th percetile scores are around 52 and 78. Similarly, we can interpret for bachelors and below secondary categories as well.

Box Plot with Hue Dimension

Who are promoted and not promoted considering education and avg_training_score?. Draw Box Plot.

In [21]:

Out[21]: <seaborn.axisgrid.FacetGrid at 0x1c9f9837e80>



From this figure, We can understand 5 types of percentile scores of candidates who are promoted or not with various education levels. Candidates with master degree and avg training score value of 74 approx have been promoted in the past.

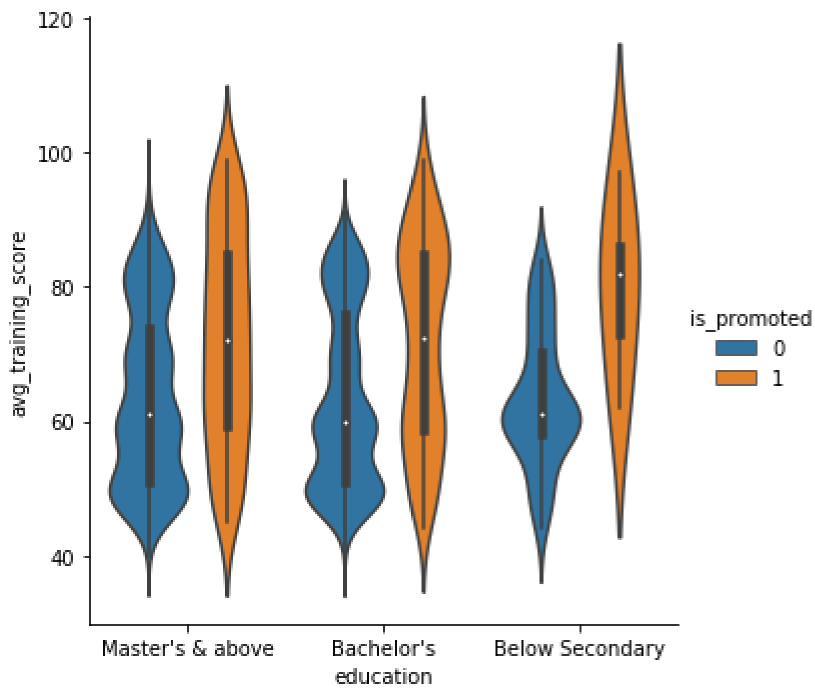
Violin Plot

The violin plots combine the boxplot and kernel density estimation procedure to provide richer description of the distribution of values. The quartile values are displayed inside the violin.

Show violin plot between education categories and avg training score with hue as "is_promoted" target variable

In [22]:

Out[22]: <seaborn.axisgrid.FacetGrid at 0x1c9f9bb5160>

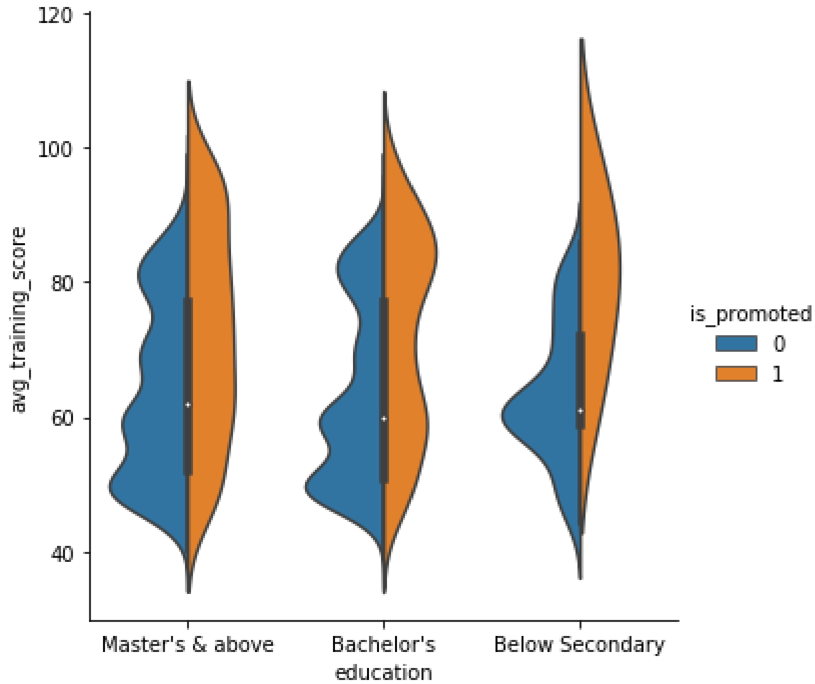


We can see in the above violin plot that each education category is represented with 2 violins one for promoted and the other not promoted target. We can also split the violin when the hue semantic parameter has only two levels, which could also be helpful in saving space on the plot.

Draw Violin plot with only 2 hue levels, use split attribute

In [23]:

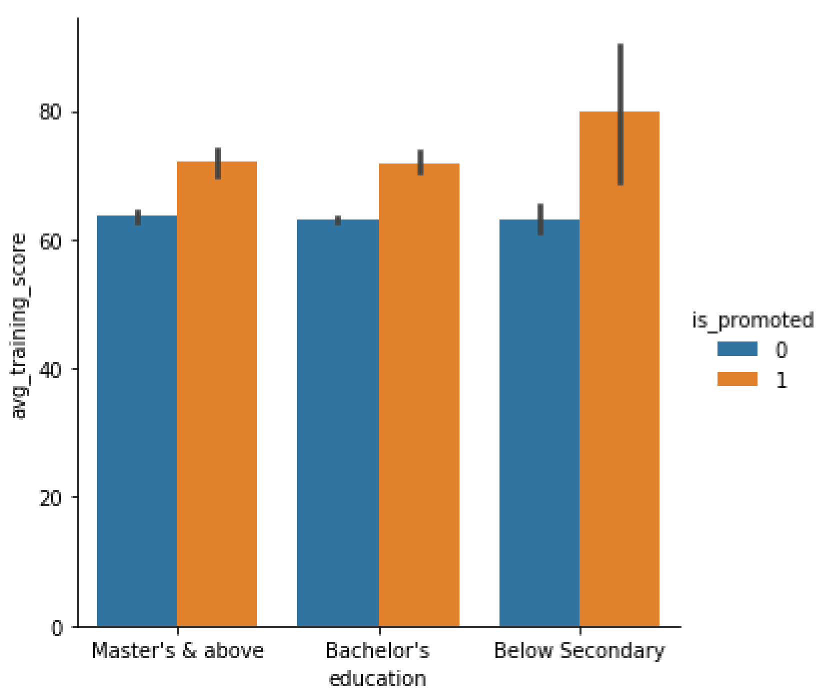
Out[23]: <seaborn.axisgrid.FacetGrid at 0x1c9fced0ef0>



Using catplot(), draw a Bar Chart between "education" and "avg_training_score", with hue as "is_promoted"

In [24]:

Out[24]: <seaborn.axisgrid.FacetGrid at 0x1c9fcedf0b8>



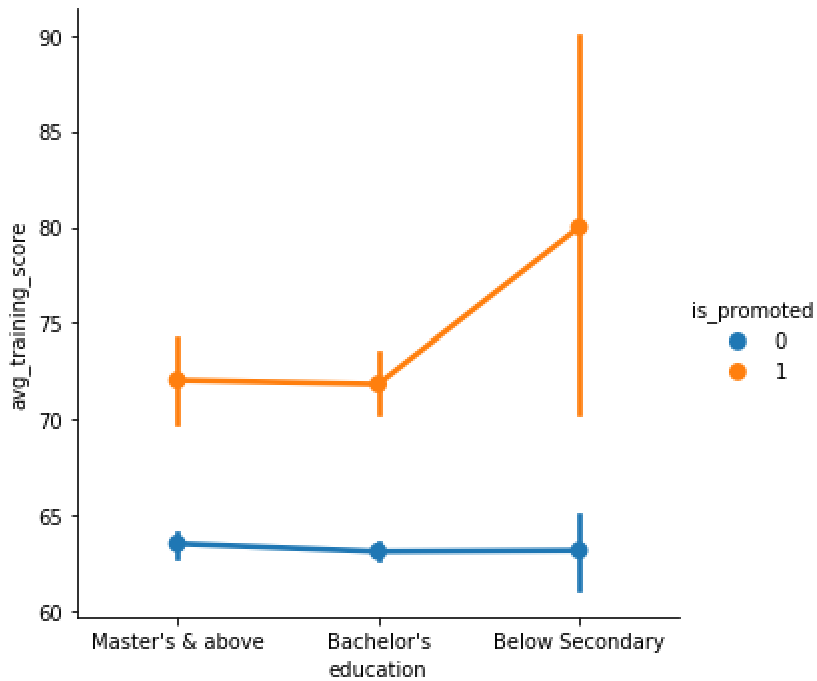
Point Plot

Point plot points out the estimate value and confidence interval. Pointplot connects data from the same hue category. This helps to identify how the relationship is changing in a particular hue category.

Show point plot between education and avg training score with hue promotion category

In [26]:

Out[26]: `<seaborn.axisgrid.FacetGrid at 0x1c9fd06da90>`



In the above figure, candidates with higher average training score are promoted. Since, we have taken mini dataset with around 700 samples, confidence interval is high for below secondary education level. Graph will show better plot if we take full dataset.

Multiple Dimension in Seaborn

So far, we have introduced 3 dimensions. Now, let us introduce another dimension, gender, in our plot. We can use Swarm plot to represent is_promoted attribute as hue and gender attribute as a faceting variable.

Draw swarm plot for education, avg training score, hue as is_promoted for male and female category

In [27]:

3. Visualizing the Distribution of Data

We want to know how data or variables are being distributed. Distribution of data could tell us a lot about the nature of the data. Types of distributions are:

```
- Univariate distribution (involves one variable)
- Bivariate distribution (involves two variables)
```

Types of plots

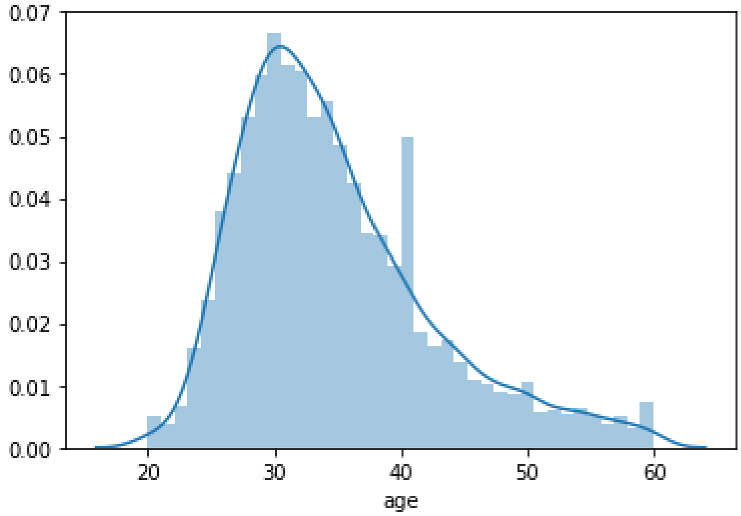
```
- For univariate distributions: Histogram
- For bivariate distributions: Hexplot, KDE plot, Boxen plot
- Correlation among all columns: heatmap
- Multiple bivariate distributions: pairplot
```

Plot Univariate Distributions

Plot Histogram with kernel density estimate value for age attribute

In [28]:

Out[28]: `<matplotlib.axes._subplots.AxesSubplot at 0x1c9fd1fbbe0>`

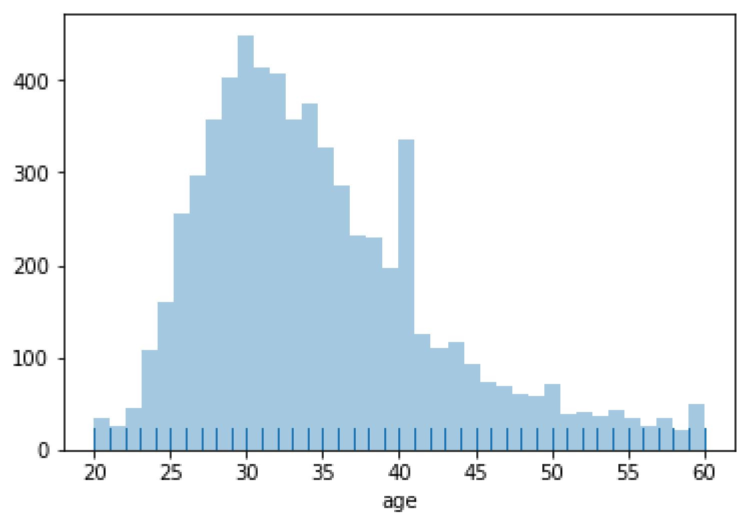


We can understand from .this plot, the average age of candidates. Most of the promotion candidates have age around 25 to 35 years.KDE plot encodes the density of observations (ie., age) on one axis with height along the other axis.

Show only Histogram for age variable, without KDE

In [29]:

Out[29]: `<matplotlib.axes._subplots.AxesSubplot at 0x1c9fd287160>`



Plot Bivariate Distributions

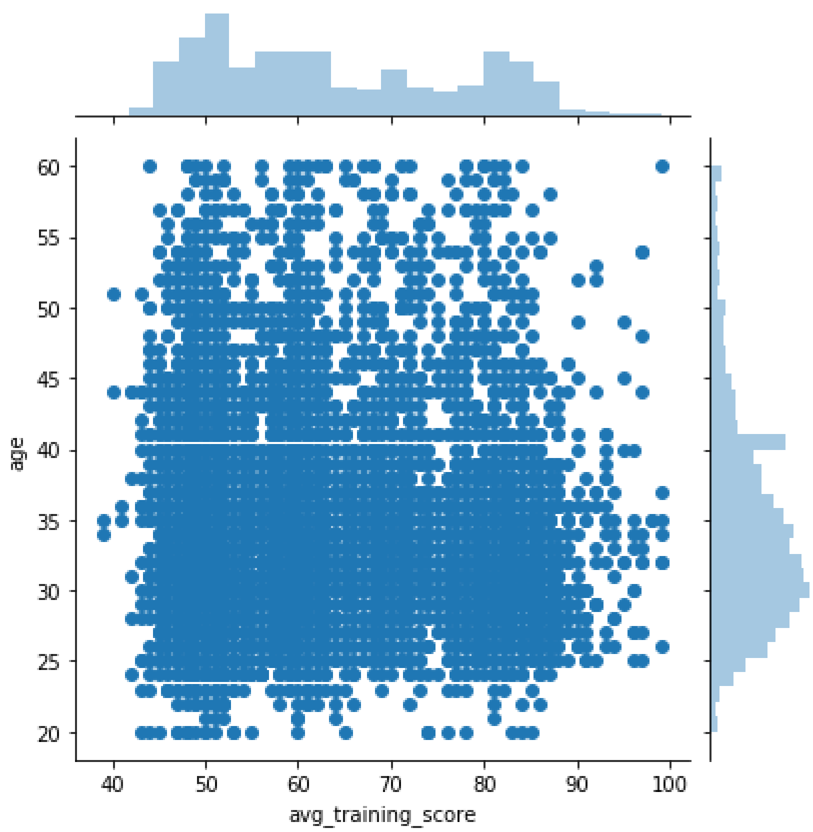
Joint Plot

We can see how two independent variables are distributed with respect to each other

Draw a joint plot between avg_training_score and age

In [30]:

Out[30]: <seaborn.axisgrid.JointGrid at 0x1c9fd2c8630>



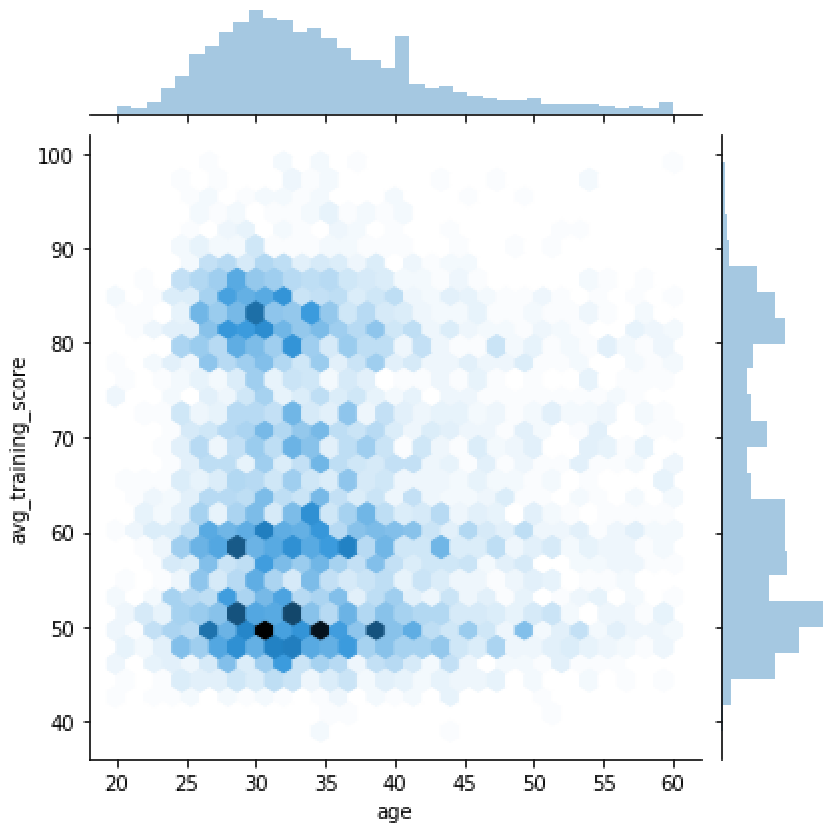
Hex Plot

Hexplot is a bivariate analog of histogram as it shows the number of observations that falls within hexagonal bins. Hexagonal binning is used in bivariate data analysis when the data is sparse in density i.e., when the data is very scattered and difficult to analyze through scatterplots

Draw a hexplot for depicting the relationship between avg training score and age

In [31]:

Out[31]: <seaborn.axisgrid.JointGrid at 0x1c9fd7bee48>



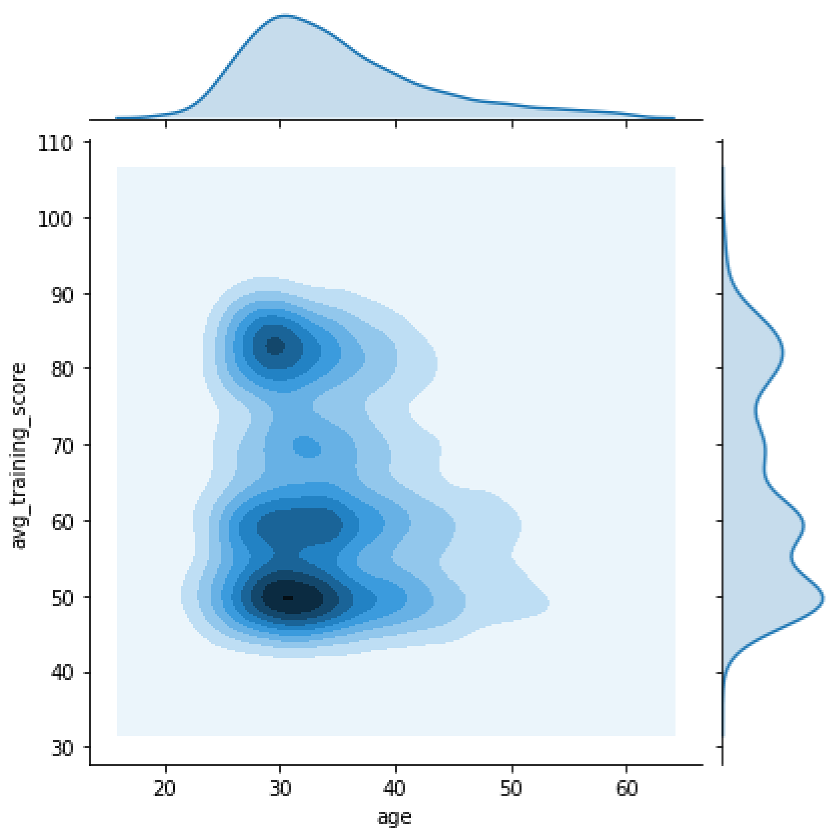
KDE Plot

It is also possible to use the kernel density estimation procedure to visualize a bivariate distribution. In seaborn, this kind of plot is shown with a contour plot and is available as a style in jointplot()to visualize the bivariate distribution.

Show KDE Plot to visualize age vs avg training score

In [33]:

Out[33]: <seaborn.axisgrid.JointGrid at 0x1c9fda503c8>



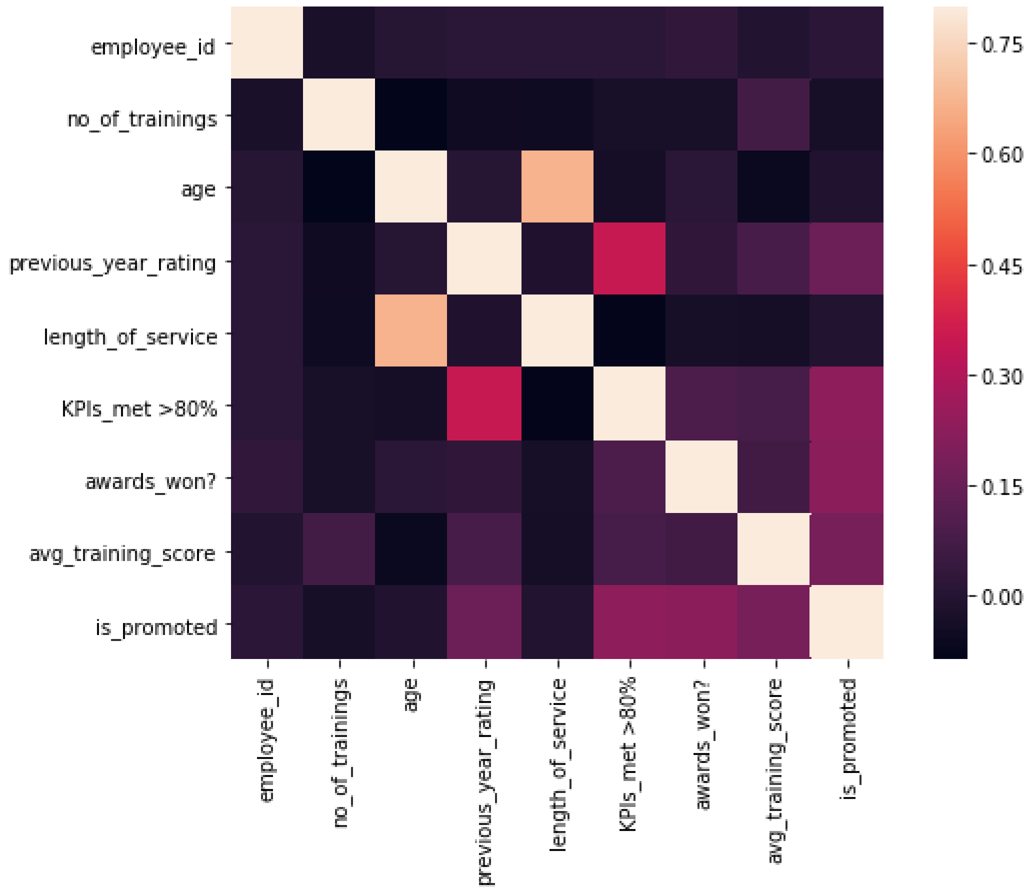
Heat Map

If you have a dataset with many columns, a good way to quickly check correlations among columns is by visualizing the correlation matrix as a heatmap.The stronger the color, the larger the correlation magnitude between columns.

Draw heatmap for the dataset

In [34]:

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1c9fdb61be0>



Can you answer these questions about the previous heatmap?

```
    - What’s the strongest and what’s the weakest correlated pair (except the main d
iagonal)?


    - What are the three variables most correlated with the target variable, is_prom
oted ?
```

Boxen Plot

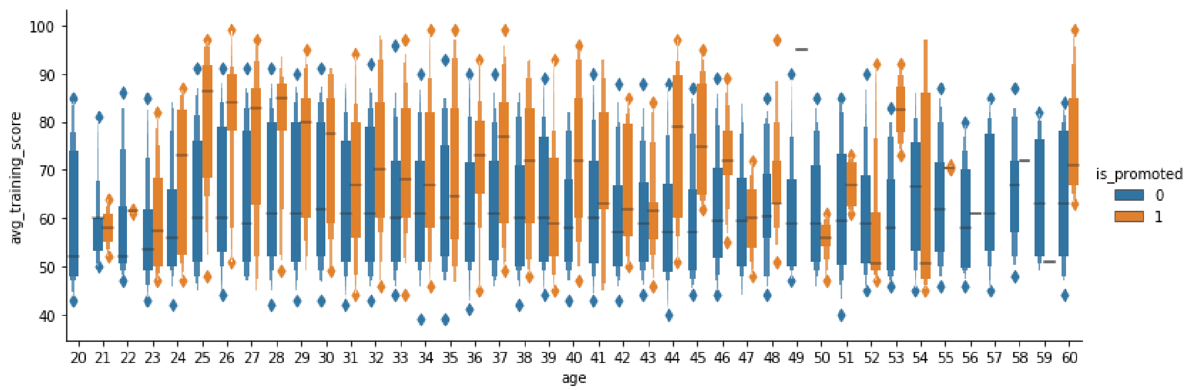
Boxen plots is used to to show the bivariate distribution. It shows large number of values of a variable, also known as quantiles. These quantiles are also defined as letter values. By plotting a large number of quantiles, it provides more insights about the shape of the distribution.

Draw Boxen Plot between "age" and "avg_training_score, with hue "is_promoted"

Adjust height and aspect values to make chart pretty

In [35]:

Out[35]: <seaborn.axisgrid.FacetGrid at 0x1c9fdc0e9b0>



Pair Plot

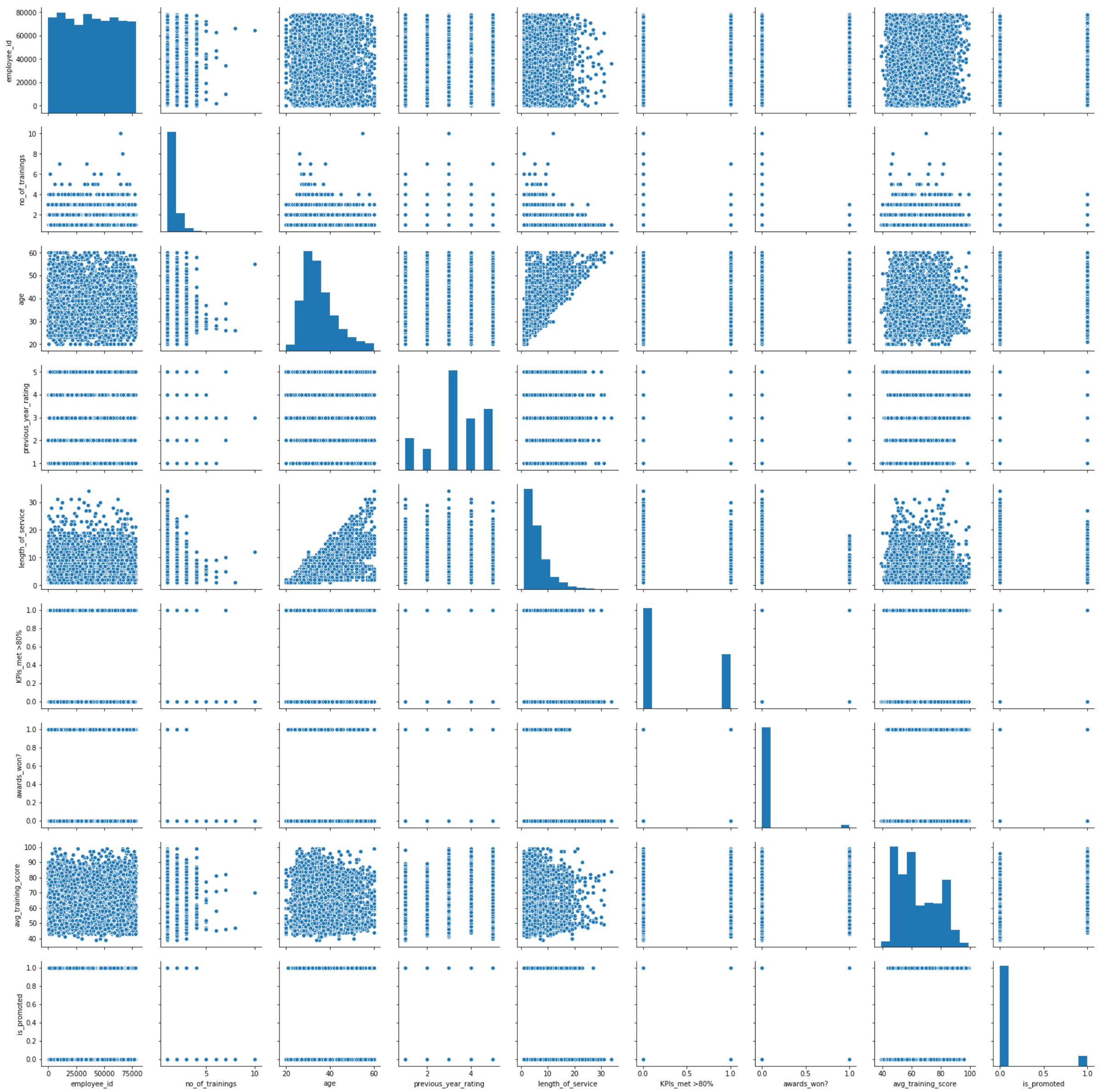
We can also plot multiple bivariate distributions in a dataset by using pairplot() function of the seaborn library. This shows the relationship between each column of the database. It also draws the univariate distribution plot of each variable on the diagonal axis

Draw a Pair Plot for the dataset

In [36]:

```
C:\Users\Rajkumar\Anaconda3\lib\site-packages\numpy\lib\histograms.py:839: Ru
ntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
C:\Users\Rajkumar\Anaconda3\lib\site-packages\numpy\lib\histograms.py:840: Ru
ntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

Out[36]: <seaborn.axisgrid.PairGrid at 0x1c9fdb61dd8>



In [ ]:

Department of Data Science - Data and Visual Analytics Lab

Lab8. Pandas Time Series Analysis

Objectives

After completing this lab, you will be able to

```
- set index with specific column
- resample a specific column or entire dataframe
- shift data forward and backward
- shift time index with day, month, year and so forth
- compute rolling window mean
- Create time series charts
```

In [1]:
```
# Importing required modules
```

In [2]:
```
# Settings for pretty plots
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
plt.show()
```

In [3]:
```
# Reading in the data
data = pd.read_csv('amazon_stock.csv')
```

Inspect top 10 rows

In [4]:
```
```

Out[4]:

	None	ticker	Date	Open	High	Low	Close	Volume	Adj_Close
0	0	AMZN	3/27/2018	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
1	1	AMZN	3/26/2018	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2	2	AMZN	3/23/2018	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
3	3	AMZN	3/22/2018	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
4	4	AMZN	3/21/2018	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Remove unwanted columns

Remove first two columns (None and ticker) as they don't add any value to the dataset. Then, print head() to check if removed

In [5]:

Out[5]:

	Date	Open	High	Low	Close	Volume	Adj_Close
0	3/27/2018	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
1	3/26/2018	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2	3/23/2018	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
3	3/22/2018	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
4	3/21/2018	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

In [6]:

```
#Look at the datatypes of the various columns, call info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1316 entries, 0 to 1315
Data columns (total 7 columns):
Date         1316 non-null object
Open         1316 non-null float64
High         1316 non-null float64
Low          1316 non-null float64
Close        1316 non-null float64
Volume       1316 non-null int64
Adj_Close    1316 non-null float64
dtypes: float64(5), int64(1), object(1)
memory usage: 72.0+ KB
```

Inspect the datatypes of columns

Looking at the information, it appears that Date column is being treated as a string rather than as dates. To fix this, we'll use the pandas to_datetime() feature which converts the arguments to dates.

Convert "Date" string column into actual Date object

In [7]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1316 entries, 0 to 1315
Data columns (total 7 columns):
Date         1316 non-null datetime64[ns]
Open         1316 non-null float64
High         1316 non-null float64
Low          1316 non-null float64
Close        1316 non-null float64
Volume       1316 non-null int64
Adj_Close    1316 non-null float64
dtypes: datetime64[ns](1), float64(5), int64(1)
memory usage: 72.0 KB
```

Let us check our data once again, with head()

In [8]:

Out[8]:

	Date	Open	High	Low	Close	Volume	Adj_Close
0	2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
1	2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2	2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
3	2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
4	2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Set Date object to be index

Here Date is one of the columns. But we want date to be the index. So, set Date as index for the data frame. Make inplace=True

In [9]:

In [10]:
```
# Check with head()
```

Out[10]:

	Open	High	Low	Close	Volume	Adj_Close
Date						
2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Understand Stock Data

Now our data has been converted into the desired format, let's take a look at its columns for further analysis.

```
- The Open and Close columns indicate the opening and closing price of the stocks o
n a particular day.
- The High and Low columns provide the highest and the lowest price for the stock o
n a particular day, respectively.
- The Volume column tells us the total volume of stocks traded on a particular day.
```

The Adj_Close column represents the adjusted closing price, or the stock's closing price on any given day of trading, amended to include any distributions and/or corporate actions occurring any time before the next day's open. The adjusted closing price is often used when examining or performing a detailed analysis of historical returns.

In [11]:
```
data['Adj_Close'].plot(figsize=(12,6),title='Adjusted Closing Price')
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x202737f5c50>

Interestingly, it appears that Amazon had a more or less steady increase in its stock price over the 2013-2018 window.

Understand DateTimeIndex

Introduction to datetime module

Python's basic tools for working with dates and times reside in the built-in datetime module. In pandas, a single point in time is represented as a pandas.Timestamp and we can use the datetime() function to create datetime objects from strings in a wide variety of date/time formats. datetimes are interchangeable with pandas.Timestamp

In [12]:
```
from datetime import datetime

my_year = 2020
my_month = 5
my_day = 1
my_hour = 13
my_minute = 36
my_second = 45

test_date = datetime(my_year, my_month, my_day)
test_date
```

Out[12]: datetime.datetime(2020, 5, 1, 0, 0) test_date = datetime(my_year, my_month, my_day, my_hour, my_minute, my_second) print("The day is : ", test_date.day) print("The hour is : ", test_date.hour) print("The month is : ", test_date.month)

Find minimum and maximum dates from data frame, call info() method

In [13]:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1316 entries, 2018-03-27 to 2013-01-02
Data columns (total 6 columns):
Open         1316 non-null float64
High         1316 non-null float64
Low          1316 non-null float64
Close        1316 non-null float64
Volume       1316 non-null int64
Adj_Close    1316 non-null float64
dtypes: float64(5), int64(1)
memory usage: 72.0 KB
```

For our stock price dataset, the type of the index column is DatetimeIndex. We can use pandas to obtain the minimum and maximum dates in the data.

Print minimum and maximum index value of dataframe

In [14]:

```
2018-03-27 00:00:00
2013-01-02 00:00:00
```

Retrieve index of earliest and latest dates using argmin and argmax

We can also calculate the latest date location and the earliest date index location as follows

In [15]:

Out[15]: 1315

In [16]:

Out[16]: 0

1.Resampling Operation

Resample entire data frame

Examining stock price data for every single day isn’t of much use to financial institutions, who are more interested in spotting market trends. To make it easier, we use a process called time resampling to aggregate data into a defined time period, such as by month or by quarter. Institutions can then see an overview of stock prices and make decisions according to these trends.

Resample data with year end frequency ("Y") with average stock price

In [17]:

Out[17]:

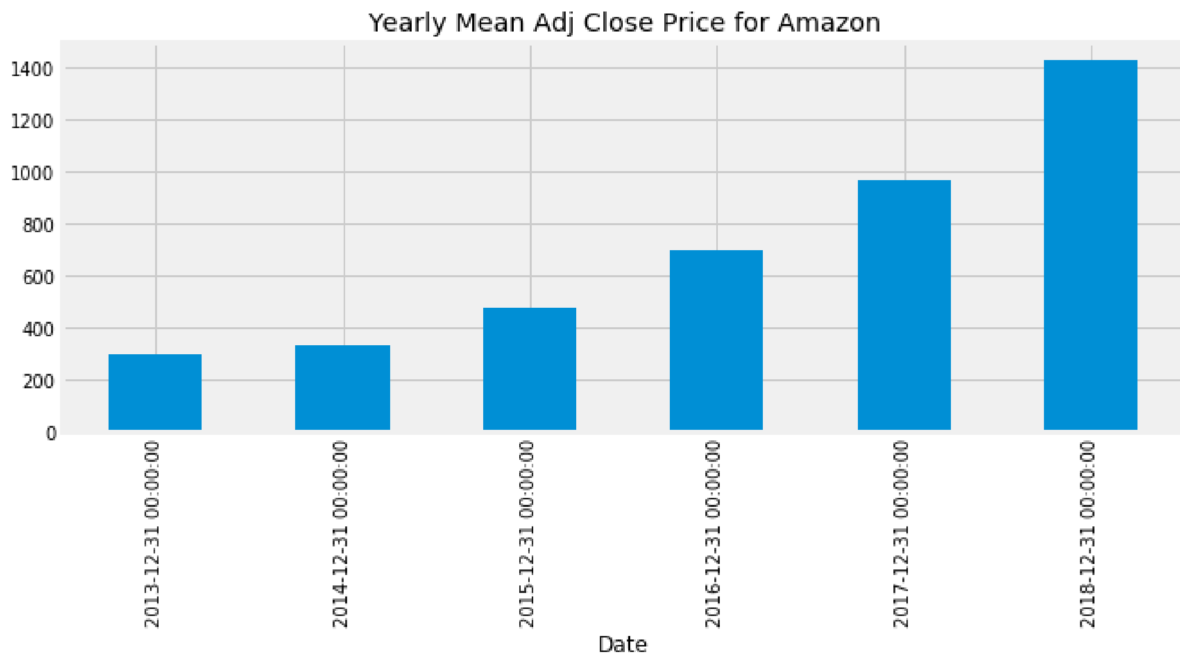
Date	Open	High	Low	Close	Volume	Adj_Close
2013-12-31	297.877223	300.925966	294.656658	298.032235	2.967880e+06	298.032235
2014-12-31	332.798433	336.317462	328.545440	332.550976	4.083223e+06	332.550976
2015-12-31	478.126230	483.248272	472.875443	478.137321	3.797801e+06	478.137321
2016-12-31	699.669762	705.799103	692.646189	699.523135	4.122043e+06	699.523135
2017-12-31	967.565060	973.789752	959.991826	967.403996	3.466207e+06	967.403996
2018-12-31	1429.770000	1446.701017	1409.469661	1429.991186	5.586829e+06	1429.991186

Here, average stock data displayed for December 31st of every year. To find other offset values refer Pandas documentation.

Resample a specific column

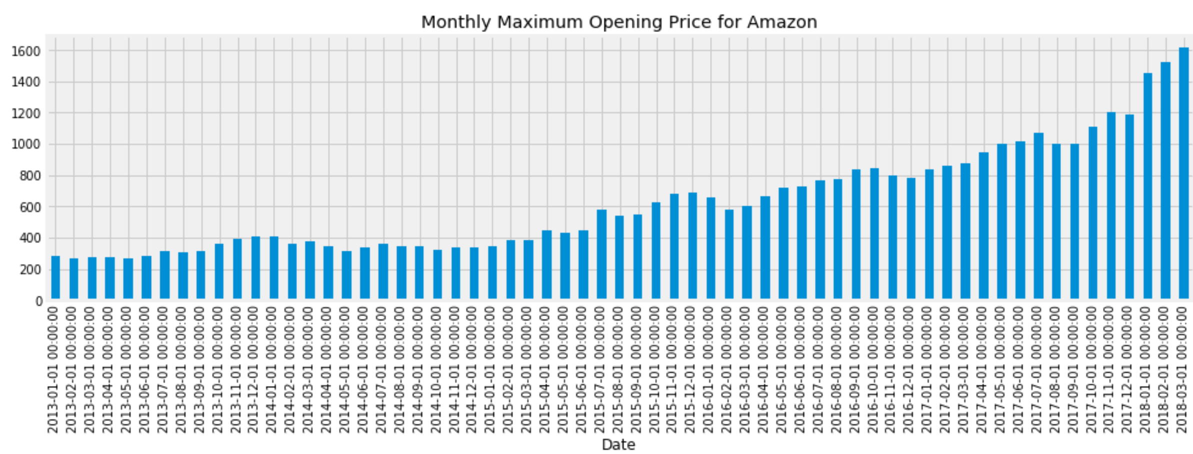
Plot a bar chart to show the yearly (Use "A") mean adjusted close price

```
In [18]: data['Adj_Close'].resample('A').mean().plot(kind='bar', figsize=(10, 4))
         plt.title('Yearly Mean Adj Close Price for Amazon')
         plt.show()
```



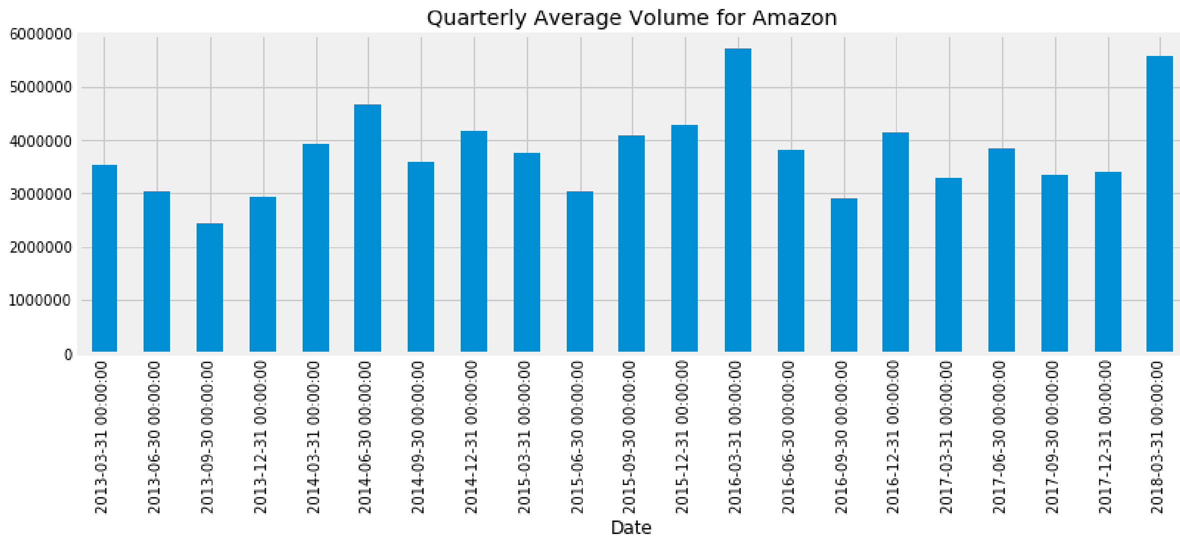
Plot bar chart to show monthly maximum (Use "MS") opening price for all years

```
In [19]:
```



Plot bar chart of Quarterly (Use "Q") Average Volume for all years

In [20]:



2.Time Shifting Operations

Shifting data forward and backward

Show head of data

In [21]:

Out[21]:

	Open	High	Low	Close	Volume	Adj_Close
Date						
2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Shift data by 1 Day forward

In [22]:

Out[22]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-26	1572.40	1575.96	1482.32	1497.05	6793279.0	1497.05
2018-03-23	1530.00	1556.99	1499.25	1555.86	5547618.0	1555.86
2018-03-22	1539.01	1549.02	1495.36	1495.56	7843966.0	1495.56
2018-03-21	1565.47	1573.85	1542.40	1544.10	6177737.0	1544.10

Shift data by 1 Day Backward

In [23]:

Out[23]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	1530.00	1556.99	1499.25	1555.86	5547618.0	1555.86
2018-03-26	1539.01	1549.02	1495.36	1495.56	7843966.0	1495.56
2018-03-23	1565.47	1573.85	1542.40	1544.10	6177737.0	1544.10
2018-03-22	1586.45	1590.00	1563.17	1581.86	4667291.0	1581.86
2018-03-21	1550.34	1587.00	1545.41	1586.51	4507049.0	1586.51

Shifting Time Index

In [24]:
```
data.head(10)
```

Out[24]:

Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-27	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-26	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-23	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-22	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-21	1586.45	1590.00	1563.17	1581.86	4667291	1581.86
2018-03-20	1550.34	1587.00	1545.41	1586.51	4507049	1586.51
2018-03-19	1554.53	1561.66	1525.35	1544.93	6376619	1544.93
2018-03-16	1583.45	1589.44	1567.50	1571.68	5145054	1571.68
2018-03-15	1595.00	1596.91	1578.11	1582.32	4026744	1582.32
2018-03-14	1597.00	1606.44	1590.89	1591.00	4164395	1591.00

Shift Time Index by 3 Months

In [25]:
```

```

Out[25]:

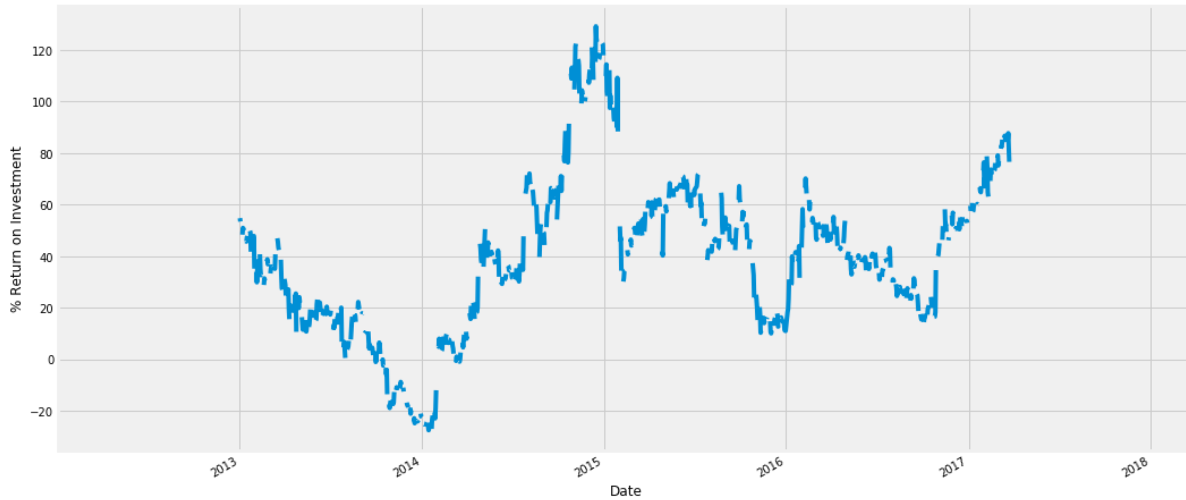
Date	Open	High	Low	Close	Volume	Adj_Close
2018-03-28	1572.40	1575.96	1482.32	1497.05	6793279	1497.05
2018-03-27	1530.00	1556.99	1499.25	1555.86	5547618	1555.86
2018-03-24	1539.01	1549.02	1495.36	1495.56	7843966	1495.56
2018-03-23	1565.47	1573.85	1542.40	1544.10	6177737	1544.10
2018-03-22	1586.45	1590.00	1563.17	1581.86	4667291	1581.86

Application: Computing Return on investment

A common context for this type of shift is computing differences over time. For example, we use shifted values to compute the one-year return on investment for Amazon stock over the course of the dataset

In [26]:
```
ROI = 100 * (data['Adj_Close'].tshift(periods=-365, freq = 'D') / data['Adj_Close'] - 1)
ROI.plot(figsize=(16,8))
plt.ylabel('% Return on Investment')
```

Out[26]: `Text(0, 0.5, '% Return on Investment')`



3. Rolling Window or Moving Window Operations

Time series data can be noisy due to high fluctuations in the market. As a result, it becomes difficult to gauge a trend or pattern in the data. Here is a visualization of the Amazon's adjusted close price over the years where we can see such noise (ie, line is not smooth).

In [27]:
```
data['Adj_Close'].plot(figsize = (12,8), color='red')
```

Out[27]: `<matplotlib.axes._subplots.AxesSubplot at 0x202759270b8>`

It would be nice if we could average this out by a week, which is where a rolling mean comes in. A rolling mean, or moving average, is a transformation method which helps average out noise from data. It works by simply splitting and aggregating the data into windows according to function, such as mean(), median(), count(), etc.

Find rolling mean for 7 days and show top-10 rows

In [28]:

Out[28]:

	Open	High	Low	Close	Volume	Adj_Close
Date						
2018-03-27	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-26	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-23	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-22	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-21	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-20	NaN	NaN	NaN	NaN	NaN	NaN
2018-03-19	1556.885714	1570.640000	1521.894286	1543.695714	5.987651e+06	1543.695714
2018-03-16	1558.464286	1572.565714	1534.062857	1554.357143	5.752191e+06	1554.357143
2018-03-15	1567.750000	1578.268571	1545.328571	1558.137143	5.534923e+06	1558.137143
2018-03-14	1576.034286	1586.471429	1558.975714	1571.771429	5.009270e+06	1571.771429

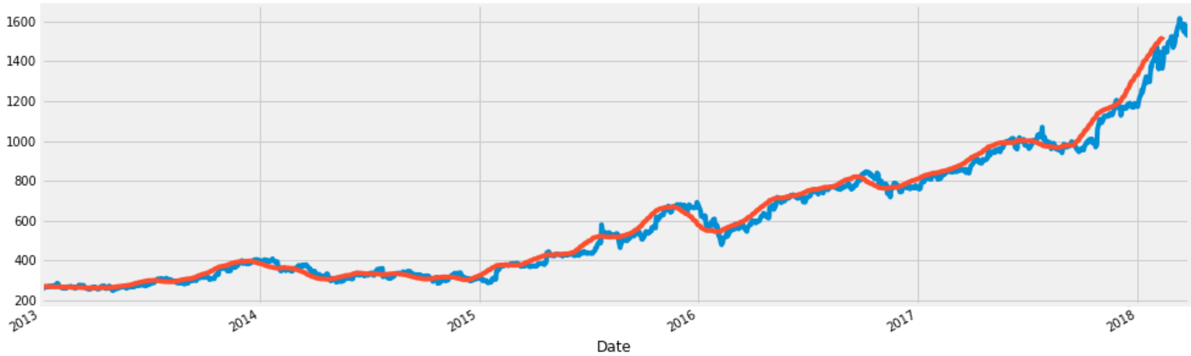
The first six values have all become blank as there wasn't enough data to actually fill them when using a window of seven days

Plot a line char for "Open" column.

Followed by, average rolling window of 30 days on the same "Open" column

In [29]:

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x20275c220b8>



Remember, first 29 days aren’t going to have the blue line because there wasn’t enough data to actually calculate that rolling mean.

In [ ]:

Department of Data Science - Data and Visual Analytics Lab

Lab9. EDA on Cardiovascular Data

Objectives

In this lab, you will perform Exploratory Data Analysis on Cardiovascular data.

```
- You will understand the features of the dataset, its size, shape, basic informati
on and datatypes of each feature.

- Then you will perform data cleaning, data wrangling and data visualization on the
dataset.

- Further, you will answer several questions about a dataset on cardiovascular dise
ase by writing code in Pandas and visualization.
```

The machine learning problem requires to predict the presence or absence of cardiovascular disease (CVD) using the patient examination results, which is beyond the scope of your course. You will simply perform EDA on the dataset.

Dataset Description

```
age int (days)
height  int (cm)
weight  float (kg)
gender  categorical code  # 1-male, 2-female
ap_hi   int  # Systolic blood pressure
ap_lo   int  # Diastolic blood pressure
cholesterol 1: normal, 2: above normal, 3: well above normal
gluc    1: normal, 2: above normal, 3: well above normal
smoke   binary  # smoking or not, 0-no, 1-yes
alco    binary # alcohol intake or not
active  binary  # physically active or not
cardio  binary  # presence or absence of cardiovascular discese
```

Import necessary packages

In [1]:
```
# import all required modules



# Disable warnings



# Import plotting modules



# import statistical module
```

Import dataset into DataFrame

In [2]:
```
df = pd.read_csv("mlbootcamp5_train.csv", sep=';')
df.head()
```

Out[2]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	carc
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	

Print the size

In [3]:
```

```

```
Dataset Size:  (70000, 13)
```

Count Values

How many people smoke?

In [4]:

Out[4]:
```
0    63831
1     6169
Name: smoke, dtype: int64
```

How many people consume alcohol?

In [5]:

Out[5]:
```
0    66236
1     3764
Name: alco, dtype: int64
```

What are the difference glucose levels?

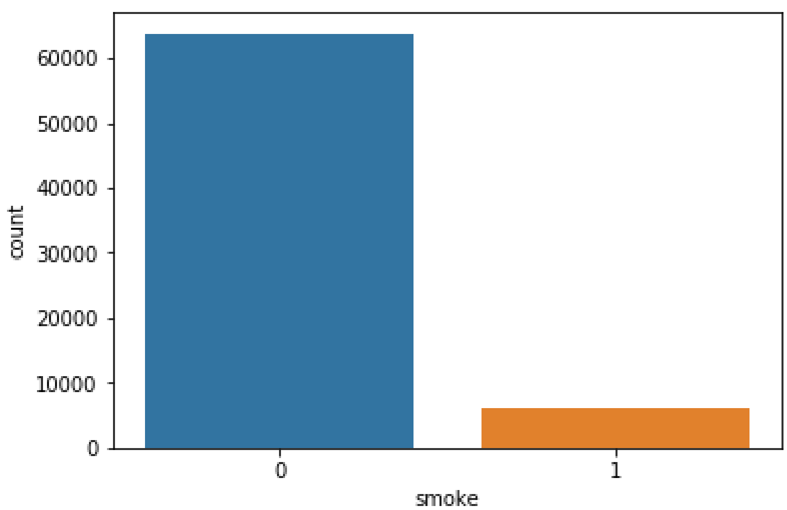
In [6]:

Out[6]:
```
1    59479
3     5331
2     5190
Name: gluc, dtype: int64
```

Draw bar chart for smoke column

In [7]:

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1458000eda0>
```



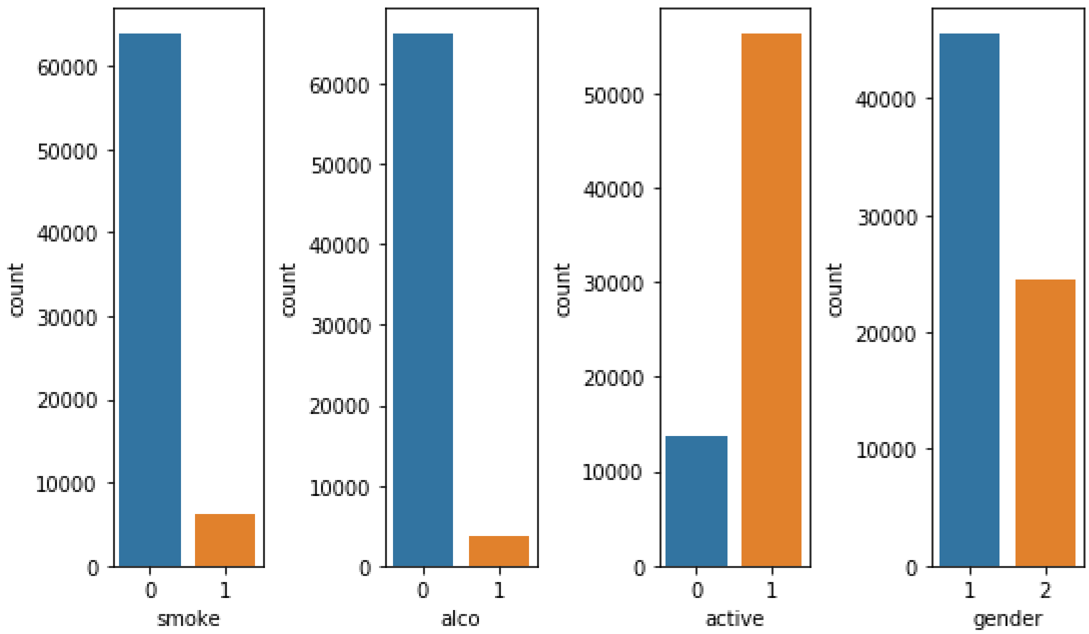
Draw 4 count plots for gender, smoke, alco and active columns respectively in 1 row, 4 columns

In [8]:
```
# First extract all 4 columns into a dataframe, binary_df

binary_df =
```

Then, plot count plots

In [9]:
```

```

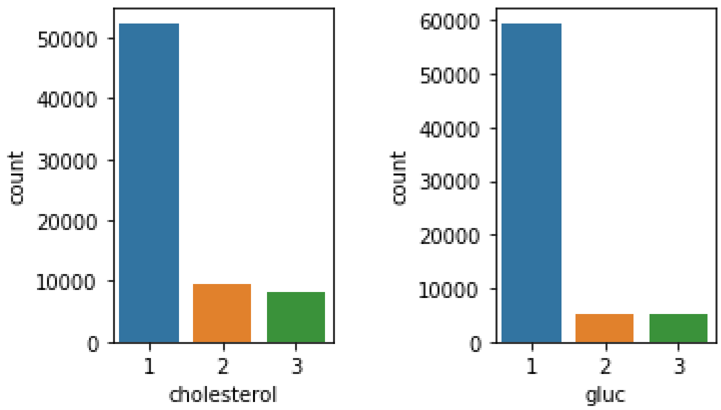
```
C:\Users\Rajkumar\Anaconda3\lib\site-packages\matplotlib\figure.py:445: UserW
arning: Matplotlib is currently using module://ipykernel.pylab.backend_inlin
e, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



Draw a count plot for cholesterol and gluc columns

In [10]:

```
C:\Users\Rajkumar\Anaconda3\lib\site-packages\matplotlib\figure.py:445: UserW
arning: Matplotlib is currently using module://ipykernel.pylab.backend_inlin
e, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```

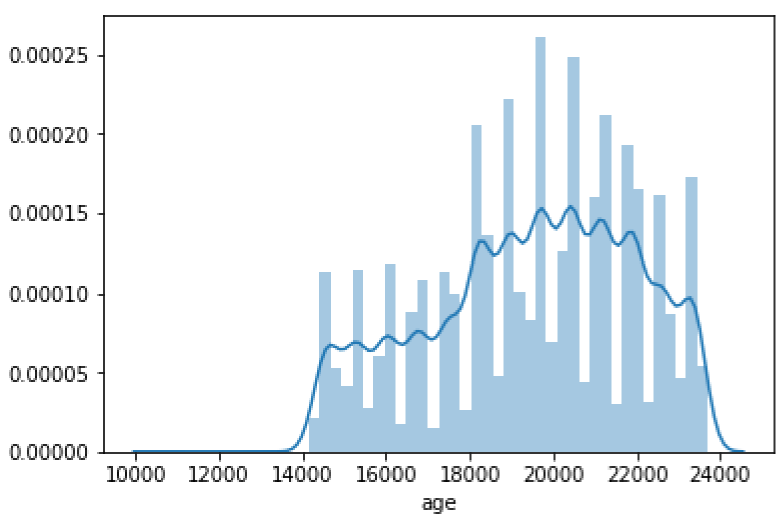


Plot Data Distribution

Show the distribution of age values as histogram

In [11]:

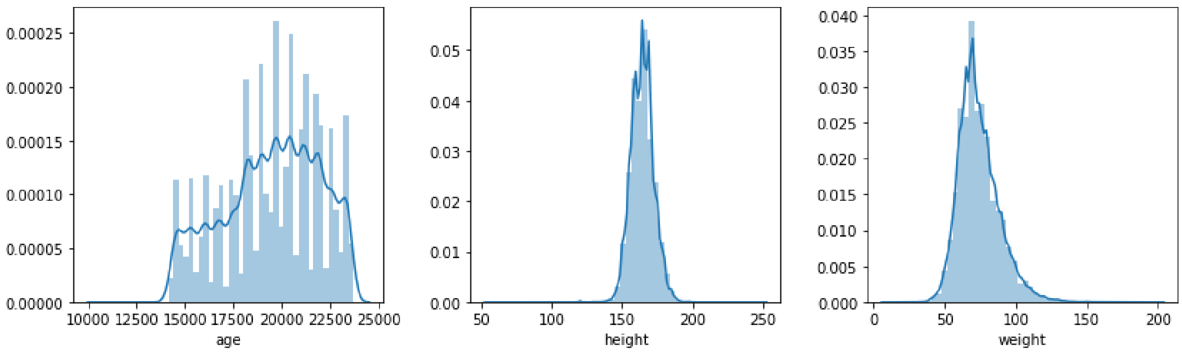
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x14582dd7f28>



Show the distribution of age, height and weight values as 3 histograms in one plot

In [12]:

```
C:\Users\Rajkumar\Anaconda3\lib\site-packages\matplotlib\figure.py:445: UserW
arning: Matplotlib is currently using module://ipykernel.pylab.backend_inlin
e, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



Calculate Summary Statistics Using Pandas

1. How many men and women are present in this dataset?

In [13]:

```
#Now, count gender column
```

Out[13]:

```
1    45530
2    24470
Name: gender, dtype: int64
```

But, we do not know if 1 means male or female. Similarly, 2 means male or female. We need to somehow find it out. How to do that?. When we inspect other columns, we can find out that there is a column "height" in centimeters. So, we can assume that men are more taller than women, generally.

So, we can compute the average height for gender=1 and gender=2. The largest average value will denote "male".

In [14]:

Out[14]:

```
gender
1    161.355612
2    169.947895
Name: height, dtype: float64
```

161 cm and almost 170 cm on average, so we make a conclusion that gender=1 represents females, and gender=2 – males.

Therefore, looking at the value_counts() of gender column, we can conclude that the dataset contains 45530 women and 24470 men.

2. Which gender more often reports consuming alcohol - men or women?

In [15]:

Out[15]:
```
gender
1    0.025500
2    0.106375
Name: alco, dtype: float64
```

Here, larger value is 2, which denotes men

3. Which gender is more physically active - men or women?

In [16]:

Out[16]:
```
gender
1    0.802021
2    0.806906
Name: active, dtype: float64
```

Here, larger values denotes 2, so answer is men

4. What is the the rounded difference between the percentages of smokers among men and women (rounded)?

First, let us find who smokes more.

In [17]:

Out[17]:
```
gender
1    0.017856
2    0.218880
Name: smoke, dtype: float64
```

So, men smokes more tha women. Now, let us find out what percentage men smokes more than women

In [18]:

Out[18]:
```
20
```

5. What is the difference between median values of age for smokers and non-smokers (in months, rounded)? You'll need to figure out the units of feature age in this dataset

In the dataset, age is given in terms of days. Therefore, you should divide by 365 to convert age into years. First, find the median age in years of smoke category.

In [19]:

Out[19]:
```
smoke
0    53.995893
1    52.361396
Name: age, dtype: float64
```

Median age of smokers is 52.4 years, for non-smokers it's 54. We see that the correct answer is 20 months.

Now, subtract the median age to find out the difference.

In [20]:

Out[20]:
```
19.613963039014372
```

Perform Risk Analysis

Calculate a new feature, age_years

The age variable represents age in days. You need to transform each age into years rounded as integer and store in new column, age_years

In [21]:

Check age_years column using head()

In [22]:
```
df.head()
```

Out[22]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	carc
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	

What is maximum age_years?

In [23]:

Out[23]: 64

What is minimum age_years?

In [24]:

Out[24]: 29

Risk Factors for Cardio Vascular Discese

```
Men who are 50 and above
Men who are smokers
Men whose cholesterol level > 1
Men whose systolic pressure is from 160 to 180 (both inclusive)
```

How many risky men are in the dataset?

How many people who are 50 and above?

In [25]:

In [26]:
```
# Show its head()

df_old.head()
```

Out[26]:
```
0     True
1     True
2     True
3    False
4    False
Name: age_years, dtype: bool
```

Now, count its unique values

In [27]:
```

```

Out[27]:
```
True     48591
False    21409
Name: age_years, dtype: int64
```

Therefore, there are 48591 people who are 50 years and above

How many are 50 years and above and men and smokers?

In [28]:
```
df_smoke_old_men =
```

In [29]:
```
# prit top-5 from df_smoke_old_men
```

Out[29]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
19	29	21755	2	162	56.0	120	70	1	1	1	0	1
38	52	23388	2	162	72.0	130	80	1	1	1	0	1
67	90	22099	2	171	97.0	150	100	3	1	1	0	1
105	140	20627	2	168	78.0	140	90	2	1	1	0	1
121	166	19507	2	174	77.0	120	80	1	1	1	0	1

How many old men have their cholesterol level > 1 and systolic pressure is from 160 to 180 too ?

In [30]:
```
risky_men =
```

In [31]:
```
# Print its head
```

Out[31]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
230	318	23376	2	175	75.0	180	100	3	1	1	1	1
732	1032	21652	2	167	70.0	160	90	2	1	1	1	1
2786	3930	21799	2	171	94.0	160	100	2	2	1	0	1
4099	5807	19749	2	183	85.0	180	110	2	1	1	0	1
4216	5950	19063	2	175	94.0	170	110	3	3	1	0	0

What is the size of risky_men ?

In [32]:

Out[32]:
```
(130, 14)
```

Therefore, ther are 136 risky men are in the dataset

How many risky men have cardiovascular discese out of these 136 samples?

In [33]:

Out[33]:
```
True     116
False     14
Name: cardio, dtype: int64
```

Conclusion: There are 122 cardiovascular discese men in the dataset

Compute Body Mass Index

Create a new feature – BMI. To do this, divide weight in kilogramms by the square of the height in meters. Normal BMI values are said to be from 18.5 to 25.

In our dataset, height is in centimeters. So, while you are computing BMI, you have to convert into meters by dividing it by 100

Create a column bmi and store the bmi values

In [34]:

In [35]:
```
df.head()
```

Out[35]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	carc
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	

How many people have ideal BMI values?

We already know that ideal BMI values are said to be from 18.5 to 25.

Compute ideal bmi values using bmi column and store the result in a new column, ideal_bmi

In [36]:
```
ideal_bmi =
```

In [37]:
```
ideal_bmi.shape
```

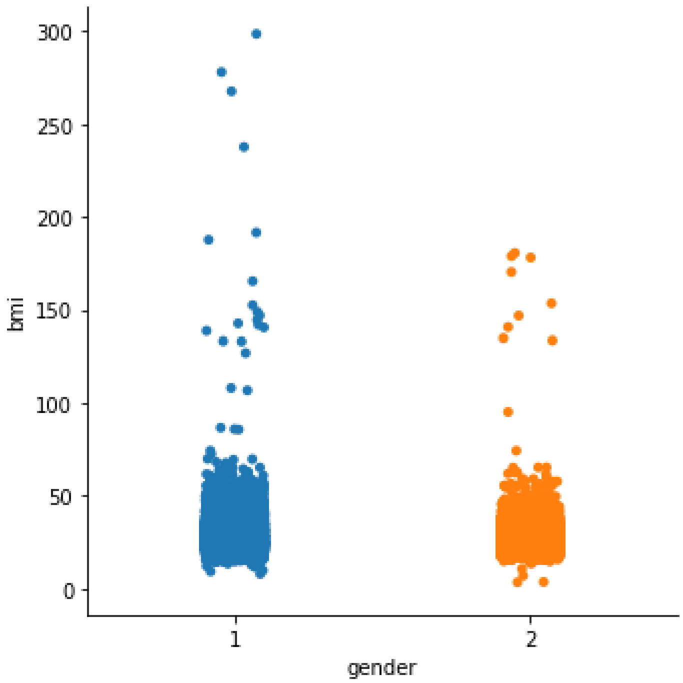
Out[37]:
```
(25804, 15)
```

25804 people have ideal BMI values

Draw catplot between gender and bmi values

In [38]:

Out[38]: <seaborn.axisgrid.FacetGrid at 0x145831a63c8>



Looking at catplot, is BMI of male is larger than BMI of female (we know 1-female, 2-male already) ?

From the plot, we can conclude Female bmi is greater than Male bmi

Is median value of Men's BMI is higher then women's BMI?

Compute median bmi for gender

In [39]:

Out[39]:
```
gender
1    26.709402
2    25.910684
Name: bmi, dtype: float64
```

From the above values, we conclude that Female have higher BMI values than male

Consider the output of the following query and answer the questions

In [40]:
```
df.groupby(['gender', 'alco', 'cardio'])['bmi'].median().to_frame()
```

Out[40]:

gender	alco	cardio	bmi
1	0	0	25.654372
		1	27.885187
	1	0	27.885187
		1	30.110991
2	0	0	25.102391
		1	26.674874
	1	0	25.351541
		1	27.530797

Is it true?. Healthy people have, on average, a higher BMI than the people with CVD.

Is it true?. For healthy, non-drinking men, BMI is closer to the norm than for healthy, non-drinking women

Data Cleaning

Remove the following people, that we consider to have erroneous data, from the dataset

```
- diastilic pressure is higher then systolic
- height is strictly less than 2.5%-percentile
- height is strictly more than 97.5%-percentile
- weight is strictly less then 2.5%-percentile
- weight is strictly more than 97.5%-percentile
```

Here, we will retain those records which do not satisfy the above conditions

In [41]:

```
filered_df = 






print(filtered_df.shape[0] / df.shape[0])
```

```
0.9037
```

So, what percentage of people do you remove from dataset?

Visual Data Analytics

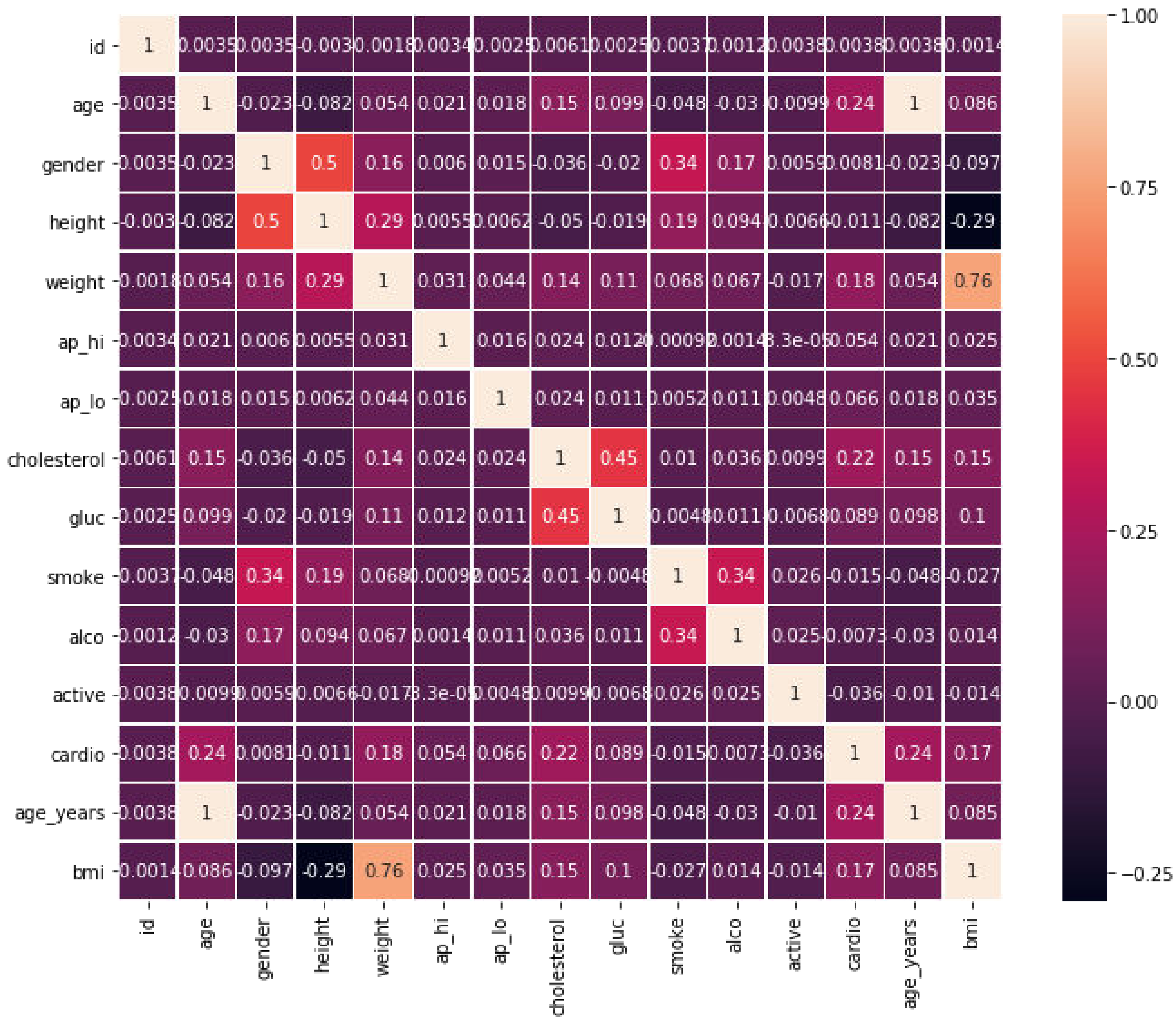
Correlation matrix visualization

To understand the features better, you can create a matrix of the correlation coefficients between the features. Use the initial dataset (non-filtered).

Plot a correlation matrix using heatmap().

In [42]:

Out[42]: `<matplotlib.axes._subplots.AxesSubplot at 0x145834951d0>`



From the Heatmap, find out top two features that have strongest Pearson's correlation with the gender feature.

In the Heatmap, which feature strongly correlates to weight?

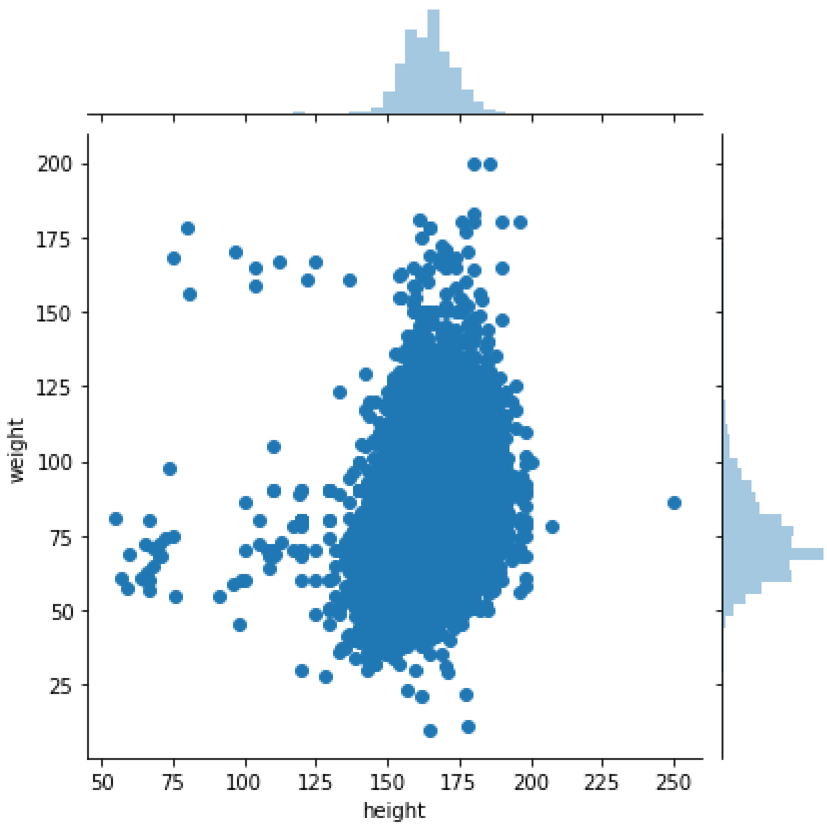
Height and Weight Distribution

Joint Plot between height and weight columns

Let us see how two independent variables, height and weight, are distributed in the dataset using Joint Plot.
Draw a Joint Plot

In [43]:

Out[43]: <seaborn.axisgrid.JointGrid at 0x14582d05080>

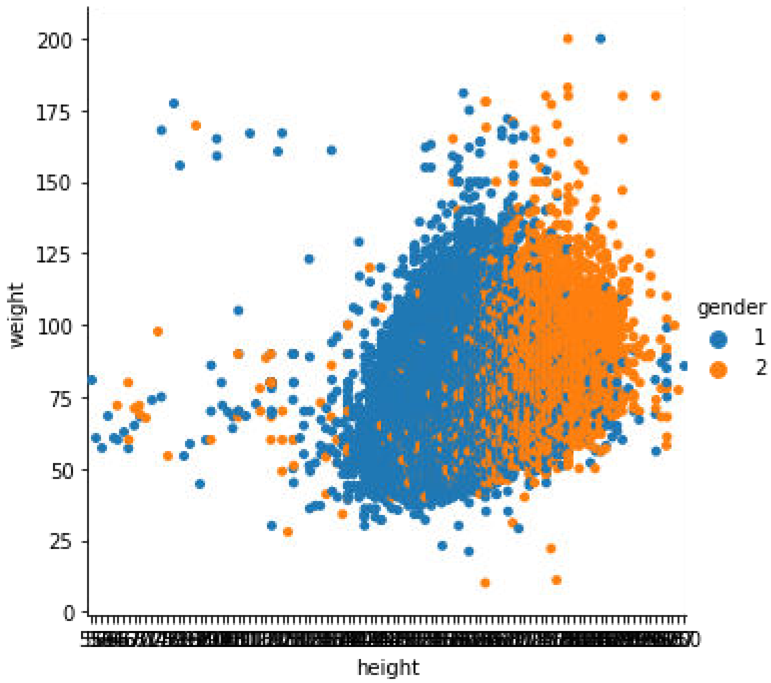


Distribution of height and weight for gender

Draw a catplot between height and weight with hue as "gender"

In [44]:

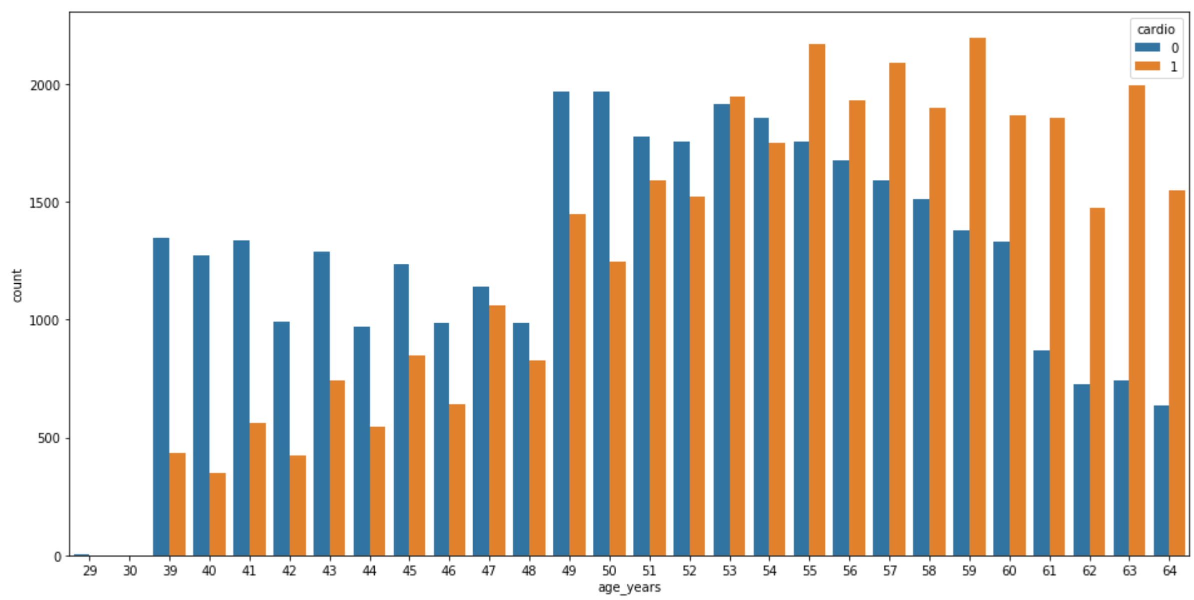
Out[44]: <seaborn.axisgrid.FacetGrid at 0x14582d5fdd8>



Find relationship between age_years and Cardio discese. Draw countplot with hue as "cardio"

In [45]:

Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x14585afdeb8>



From the above figure, we know critical age for cardio discese is between 50 and 60.

Note: You should use plt.rcParams to modify figure size.

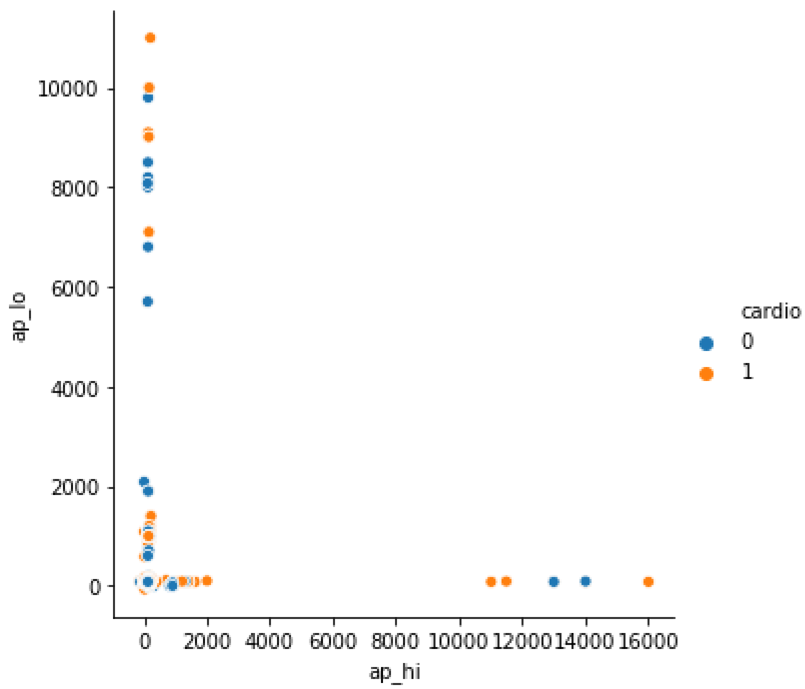
How diastilic and systolic values affect cardio patients?

Draw Boxen plot

for plotting a large number of quantiles, which provides more insights about the shape of the distribution

In [46]:

Out[46]: <seaborn.axisgrid.FacetGrid at 0x145860f4160>



Since, the range of ap_hi and ap_lo values very large, the plot appears too contensed.

Now, print max and min values and justify.

In [47]:

Out[47]: 16020

In [48]:

Out[48]: -150

In [49]:

Out[49]: 11000

In [50]:

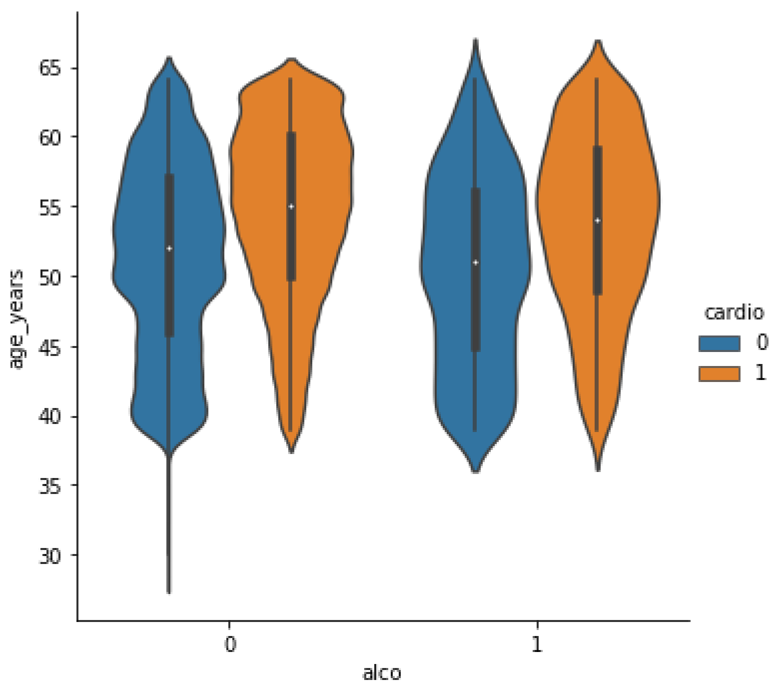
Out[50]: -70

How alcohol intake and age affect cardios?

Draw Violin Plot to represent relationship between alcohol intake and age_years with hue as "cardio"

In [51]:

Out[51]: <seaborn.axisgrid.FacetGrid at 0x145863e6278>



From this plot, we can understand the distribution of age values among alcohol consumers for cardio discese

1. For Non alcoholic category (ie., alco=0), what is the 50th percentile value for Non-Cardio (ie., cardio=0) people?

In [ ]:

2. For Non alcoholic category (ie., alco=0), what is the 50th percentile value for Cardio (ie., cardio=1) people?

In [ ]:

```

```

3. For alcoholic category (ie., alco=1), what is the 25th percentile value for Non-Cardio (ie., cardio=0) people?

In [ ]:

```

```

4. For alcoholic category (ie., alco=1), what is the 25th percentile value for Cardio (ie., cardio=1) people?

In [ ]:

```

```

Department of Data Science - Data and Visual Analytics Lab

Lab10. Advanced Data Wrangling in Pandas

Objectives ¶

After completing this lab, you will be able to create and apply some advanced features of Pandas including

```
- pivot table
- crosstab
- cut and qcut
- melt
- stack and unstack
```

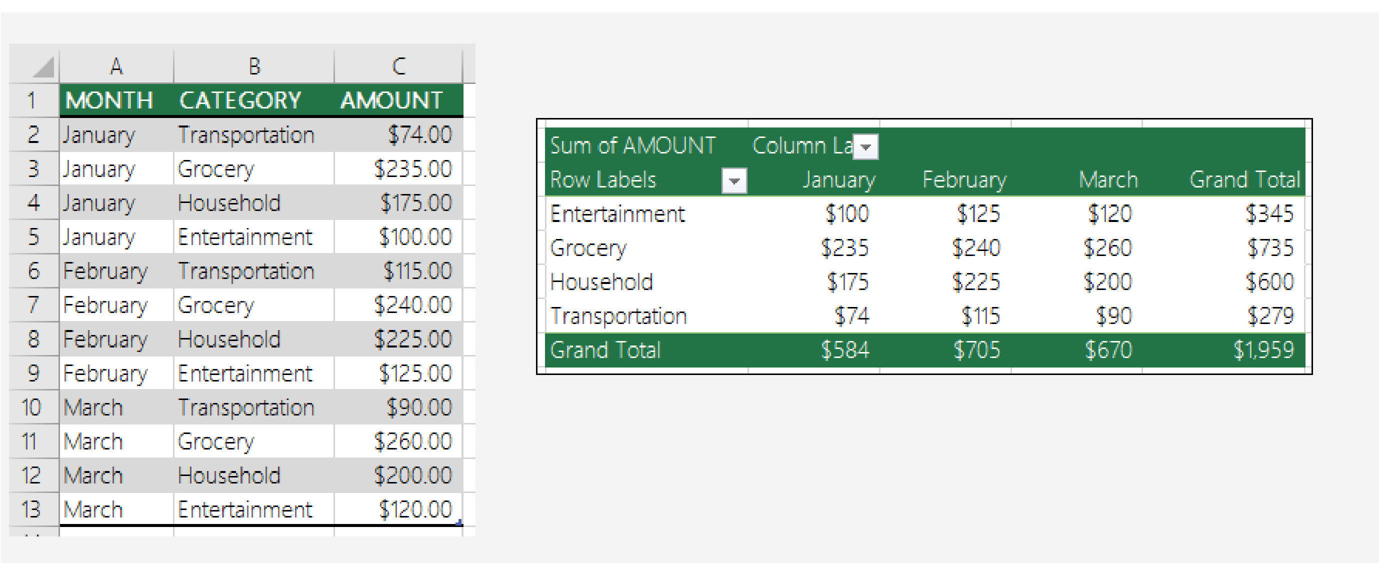
Import necessary modules

In [1]:

```

```

Pivoting data in MS Excel

People who know Excel, probably know the **Pivot** functionality:



	A	B	C
1	MONTH	CATEGORY	AMOUNT
2	January	Transportation	$74.00
3	January	Grocery	$235.00
4	January	Household	$175.00
5	January	Entertainment	$100.00
6	February	Transportation	$115.00
7	February	Grocery	$240.00
8	February	Household	$225.00
9	February	Entertainment	$125.00
10	March	Transportation	$90.00
11	March	Grocery	$260.00
12	March	Household	$200.00
13	March	Entertainment	$120.00

Sum of AMOUNT	Column La			
Row Labels	January	February	March	Grand Total
Entertainment	$100	$125	$120	$345
Grocery	$235	$240	$260	$735
Household	$175	$225	$200	$600
Transportation	$74	$115	$90	$279
Grand Total	$584	$705	$670	$1,959

The data of the table:

In [2]:
```
excelample = pd.DataFrame({'Month': ["January", "January", "January", "January",
                                     "February", "February", "February", "February",
                                     "March", "March", "March", "March"],
                          'Category': ["Transportation", "Grocery", "Household", "Entertainment",
                                       "Transportation", "Grocery", "Household", "Entertainment",
                                       "Transportation", "Grocery", "Household", "Entertainment"],
                          'Amount': [74., 235., 175., 100., 115., 240., 225., 125., 90., 260., 200., 120.]})
```

In [3]:
```
excelample
```

Out[3]:

	Month	Category	Amount
0	January	Transportation	74.0
1	January	Grocery	235.0
2	January	Household	175.0
3	January	Entertainment	100.0
4	February	Transportation	115.0
5	February	Grocery	240.0
6	February	Household	225.0
7	February	Entertainment	125.0
8	March	Transportation	90.0
9	March	Grocery	260.0
10	March	Household	200.0
11	March	Entertainment	120.0

In [4]:
```
excelample_pivot = excelample.pivot(index="Category", columns="Month", values="Amount")
excelample_pivot
```

Out[4]:

Month	**February**	**January**	**March**
Category			
Entertainment	125.0	100.0	120.0
Grocery	240.0	235.0	260.0
Household	225.0	175.0	200.0
Transportation	115.0	74.0	90.0

Interested in *Grand totals*?

In [5]:
```
# sum columns
excelample_pivot.sum(axis=1)
```

Out[5]:
```
Category
Entertainment     345.0
Grocery           735.0
Household         600.0
Transportation    279.0
dtype: float64
```

In [6]:
```
# sum rows
excelample_pivot.sum(axis=0)
```

Out[6]:
```
Month
February    705.0
January     584.0
March       670.0
dtype: float64
```

Pivot is just reordering your data

Small subsample of the titanic dataset:

In [7]:
```
df = pd.DataFrame({'Fare': [7.25, 71.2833, 51.8625, 30.0708, 7.8542, 13.0],
                   'Pclass': [3, 1, 1, 2, 3, 2],
                   'Sex': ['male', 'female', 'male', 'female', 'female', 'mal
e'],
                   'Survived': [0, 1, 0, 1, 0, 1]})
```

In [8]:
```
df
```

Out[8]:

	Fare	Pclass	Sex	Survived
0	7.2500	3	male	0
1	71.2833	1	female	1
2	51.8625	1	male	0
3	30.0708	2	female	1
4	7.8542	3	female	0
5	13.0000	2	male	1

In [9]:
```
df.pivot(index='Pclass', columns='Sex', values='Fare')
```

Out[9]:

Sex	female	male
Pclass		
1	71.2833	51.8625
2	30.0708	13.0000
3	7.8542	7.2500

Exercise: Create a Pivot table with 'Survided' values for Pclass vs Sex.

In [10]:
```
```

Out[10]:

Sex	female	male
Pclass		
1	1	0
2	1	1
3	0	0

Let's now use the full Titanic Dataset

In [11]:
```
df = sns.load_dataset('titanic')  # avaiable inbuilt with seaborn
```

In [12]:
```
df.head()
```

Out[12]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

And try the same pivot (*no worries about the try-except, this is here just used to catch a loooong error*):

In [13]:
```python
try:
    df.pivot(index='sex', columns='pclass', values='fare')
except Exception as e:
    print("Exception!", e)
```

```
Exception! Index contains duplicate entries, cannot reshape
```

This does not work, because we would end up with multiple values for one cell of the resulting frame, as the error says: `duplicated` values for the columns in the selection. As an example, consider the following rows of our three columns of interest:

In [14]:
```python
df.loc[[1, 3], ["sex", 'pclass', 'fare']]
```

Out[14]:

	sex	pclass	fare
1	female	1	71.2833
3	female	1	53.1000

Since `pivot` is just restructering data, where would both values of `Fare` for the same combination of `Sex` and `Pclass` need to go?

Well, they need to be combined, according to an `aggregation` functionality, which is supported by the function `pivot_table`

NOTE:

- **Pivot** is purely restructering: a single value for each index/column combination is required.

Pivot Tables - Aggregating while Pivoting

Pivot Table is a multidimensional version of GroupBy aggregation.

In [15]:
```python
df.pivot_table(index='sex', columns='pclass', values='fare')
```

Out[15]:

pclass	1	2	3
sex			
female	106.125798	21.970121	16.118810
male	67.226127	19.741782	12.661633

REMEMBER: * By default, `pivot_table` takes the **mean** of all values that would end up into one cell. However, you can also specify other aggregation functions using the `aggfunc` keyword.

Create a Pivot table with maximum 'fare' values for 'sex' vs 'pclass' columns

In [16]:
```
df.pivot_table(index='sex', columns='pclass',
               values='fare', aggfunc='max')
```

Out[16]:

pclass	1	2	3
sex			
female	512.3292	65.0	69.55
male	512.3292	73.5	69.55

Exercise: Create a Pivot table with the count of 'fare' values for 'sex' vs 'pclass' columns

In [17]:

Out[17]:

pclass	1	2	3
sex			
female	94	76	144
male	122	108	347

REMEMBER:

- There is a shortcut function for a `pivot_table` with a `aggfunc='count'` as aggregation: `crosstab`

In [18]:
```
pd.crosstab(index=df['sex'], columns=df['pclass'])
```

Out[18]:

pclass	1	2	3
sex			
female	94	76	144
male	122	108	347

Exercise: Make a pivot table with the mean survival rates for pclass vs sex

In [19]:

Out[19]:

pclass	1	2	3
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

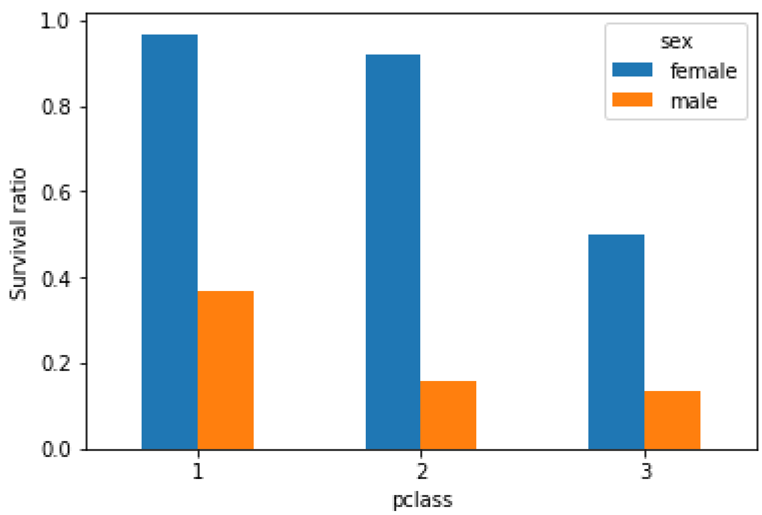
Plot Bar Chart for Survival ratio

In [20]:

```
fig, ax1 = plt.subplots()
df.pivot_table(index='pclass', columns='sex',
               values='survived', aggfunc='mean').plot(kind='bar', rot=0, ax=a
x1)
ax1.set_ylabel('Survival ratio')
```

Out[20]:

```
Text(0, 0.5, 'Survival ratio')
```



Exercise: Make a pivot table of the median Fare payed by aged vs sex

In [21]:
```
median_age_table =


# let us show only 5 rows
median_age_table[:5]
```

Out[21]:

sex	female	male
age		
0.42	NaN	8.5167
0.67	NaN	14.5000
0.75	19.25830	NaN
0.83	NaN	23.8750
0.92	NaN	151.5500
1.00	13.43750	39.0000
2.00	26.95000	27.5625
3.00	31.32710	22.3750
4.00	22.02500	29.1250
5.00	23.50415	NaN

Exercise: Make a pivot table of the median Fare payed by 'underaged' vs 'sex'

In [22]:
```
# Create a new column 'underaged' and store the result of the condition age <=
18

df['underaged'] =
```

In [23]:
```
# Now, make the pivot table for underaged
```

Out[23]:

sex	female	male
underaged		
False	24.1500	10.3354
True	20.2875	20.2500

Grouping Pivot table

```
In [24]: age = pd.cut(df['age'], [0, 18, 80])
         df.pivot_table('survived', ['sex', age], 'class')
```

Out[24]:

	class	First	Second	Third
sex	**age**			
female	**(0, 18]**	0.909091	1.000000	0.511628
	(18, 80]	0.972973	0.900000	0.423729
male	**(0, 18]**	0.800000	0.600000	0.215686
	(18, 80]	0.375000	0.071429	0.133663

We can apply this same strategy when working with the columns as well; let's add info on the fare paid using pd.qcut to automatically compute quantiles

```
In [25]: fare = pd.qcut(df['fare'], 2)
         df.pivot_table('survived', ['sex', age], [fare, 'class'])
```

Out[25]:

	fare	(-0.001, 14.454]			(14.454, 512.329]		
	class	**First**	**Second**	**Third**	**First**	**Second**	**Third**
sex	**age**						
female	**(0, 18]**	NaN	1.000000	0.714286	0.909091	1.000000	0.318182
	(18, 80]	NaN	0.880000	0.444444	0.972973	0.914286	0.391304
male	**(0, 18]**	NaN	0.000000	0.260870	0.800000	0.818182	0.178571
	(18, 80]	0.0	0.098039	0.125000	0.391304	0.030303	0.192308

The result is a four-dimensional aggregation with hierarchical indices

Multiple Aggregate Functions

```
In [26]: df.pivot_table(index='sex', columns='class',
         aggfunc={'survived':sum, 'fare':'mean'})
```

Out[26]:

	fare			survived		
class	**First**	**Second**	**Third**	**First**	**Second**	**Third**
sex						
female	106.125798	21.970121	16.118810	91	70	72
male	67.226127	19.741782	12.661633	45	17	47

Melt - from Pivot Table to long or tidy format

The `melt` function performs the inverse operation of a `pivot`. This can be used to make your frame longer, i.e. to make a *tidy* version of your data.

In [27]:
```
pivoted = df.pivot_table(index='sex', columns='pclass', values='fare').reset_i
ndex()
pivoted.columns.name = None
```

In [28]:
```
pivoted
```

Out[28]:

	sex	1	2	3
0	female	106.125798	21.970121	16.118810
1	male	67.226127	19.741782	12.661633

Assume we have a DataFrame like the above. The observations (the average Fare people payed) are spread over different columns. In a tidy dataset, each observation is stored in one row. To obtain this, we can use the `melt` function:

In [29]:
```
pd.melt(pivoted)
```

Out[29]:

	variable	value
0	sex	female
1	sex	male
2	1	106.126
3	1	67.2261
4	2	21.9701
5	2	19.7418
6	3	16.1188
7	3	12.6616

As you can see above, the `melt` function puts all column labels in one column, and all values in a second column.

In this case, this is not fully what we want. We would like to keep the 'Sex' column separately:

In [30]:
```
pd.melt(pivoted, id_vars=['sex']) #, var_name='pclass', value_name='fare')
```

Out[30]:

	sex	variable	value
0	female	1	106.125798
1	male	1	67.226127
2	female	2	21.970121
3	male	2	19.741782
4	female	3	16.118810
5	male	3	12.661633

Reshaping with `stack` and `unstack`

The docs say:

> Pivot a level of the (possibly hierarchical) column labels, returning a DataFrame (or Series in the case of an object with a single level of column labels) having a hierarchical index with a new inner-most level of row labels.

Before we speak about `hierarchical index`, first check it in practice on the following dummy example:

In [31]:
```
df2 = pd.DataFrame({'A':['one', 'one', 'two', 'two'],
                    'B':['a', 'b', 'a', 'b'],
                    'C':range(4)})
df2
```

Out[31]:

	A	B	C
0	one	a	0
1	one	b	1
2	two	a	2
3	two	b	3

To use `stack` / `unstack`, we need the values we want to shift from rows to columns or the other way around as the index:

In [32]:
```
df2 = df2.set_index(['A', 'B']) # Indeed, you can combine two indices
df2
```

Out[32]:

A	B	C
one	a	0
	b	1
two	a	2
	b	3

In [33]:
```
result = df2['C'].unstack()
result
```

Out[33]:

B (A)	a	b
one	0	1
two	2	3

In [34]:
```
df2 = result.stack().reset_index(name='C')
df2
```

Out[34]:

	A	B	C
0	one	a	0
1	one	b	1
2	two	a	2
3	two	b	3

REMEMBER:

- **stack**: make your data *longer* and *smaller*
- **unstack**: make your data *shorter* and *wider*

Mimick Pivot Table

To better understand and reason about pivot tables, we can express this method as a combination of more basic steps. In short, the pivot is a convenient way of expressing the combination of a `groupby` and `stack/unstack`.

Let us come back to our titanic dataset

In [35]:
```
df.head()
```

Out[35]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

In [36]:
```
df.pivot_table(index='pclass', columns='sex',
               values='survived', aggfunc='mean')
```

Out[36]:

sex	female	male
pclass		
1	0.968085	0.368852
2	0.921053	0.157407
3	0.500000	0.135447

Exercise:

```
- Get the same result as above based on a combination of `groupby` and `unstack`</l
i>
- First use `groupby` to calculate the survival ratio for all groups`unstack`</li>
- Then, use `unstack` to reshape the output of the groupby operation</li>
```

In [37]:
```
```

Out[37]:

sex	female	male
pclass		
1	0.968085	0.368852
2	0.921053	0.157407
3	0.500000	0.135447

In [ ]:
```
```

Department of Data Science - Data and Visual Analytics Lab

Lab11. Interactive Dash Board Creation in Tableau

Objectives

In this lab, you will create an interactive dash board using Tableau Desktop. In particular, you will learn and acquire the following skills.

```
- Creating charts
- Adding calculation to your workbook
- Mapping data in Tableau
- Interactive Dashboard Creation and Visualization
```

Tableau Projects

You can select any one of the project ideas shown below and create an interactive dash board in Tableau

Project 1: Sales Performance Analysis

Build a dashboard to present monthly sales performance by product segment and product category for the purposes of identifying the areas that have met or exceeded their sales targets

Domain: E-commerce

Project 2: Customer Analysis

Build a dashboard that presents customer statistics, ranking them by profit ratio and sales. Also, include statistics regarding profit performance by region.

Domain: Retail

Project 3: Product Analysis

Build a dashboard that presents sales by product category over time, with the ability to drill down to the product and regional level to check if the products are correctly priced.

Domain: Retail

Project 4: Sales Dashboard

Build a dashboard that presents metrics about products (e.g. sales, profits, profit ratio) and the trends of statistics over a given period of time, filtering down to a number of geographic regions.Domain:

```
In [ ]:
```