WEEK-1:

1. Demonstrate usage of branching and looping statements
2. Demonstrate Recursive functions
3. Demonstrate Lists

1. Demonstrate usage of branching and looping statements

Branching (if, elif and else)
Very often in life, and in computer programs, the next action depends on the outcome of a question starting with "if". This gives the possibility of branching into different types of action depending on some criterion.
As an introduction to branching, let us "build up" a little program that evaluates a water temperature provided by the program user.

One if test:

```
T = float(input('What is the water temperature? '))
if T > 24:
print('Great, jump in!')
# First line after if part
```

Two if-tests
Immediately, we realize that this is not satisfactory, so (as a "first fix") we extend our code with a second if test, as

```
T = float(input('What is the water temperature? '))
if T > 24: # testing condition 1
print('Great, jump in!')
if T <= 24: # testing condition 2
print('Do not swim. Too cold!')
# First line after if-if construction
```

If-else test

```
T = float(input('What is the water temperature? '))
if T > 24: # testing condition 1
print('Great, jump in!')
else:
print('Do not swim. Too cold!')
# First line after if-else construction
```

A more general form of an if-elif-else construction reads

```
if condition_1: # testing condition 1
<code line 1>
<code line 2>
...
elif condition_2: # testing condition 2
<code line 1>
<code line 2>
...
elif condition_3: # testing condition 3
<code line 1>
```

```
<code line 2>
...
else:
<code line 1>
<code line 2>
...
# First line after if-elif-else construction
```

The for Loop
Many computations are repetitive by nature and programming languages have certain *loop structures* to deal with this. One such loop structure is the *for loop*.

1.1.1 Example: Printing the 5 Times Table
Assume the task is to print out the 5 times table. Before having learned about loop structures in programming, most of us would first think of coding this like:

```
# Naively printing the 5 times table
print('{:d}*5 = {:d}'.format(1, 1*5))
print('{:d}*5 = {:d}'.format(2, 2*5))
print('{:d}*5 = {:d}'.format(3, 3*5))
print('{:d}*5 = {:d}'.format(4, 4*5))
print('{:d}*5 = {:d}'.format(5, 5*5))
print('{:d}*5 = {:d}'.format(6, 6*5))
print('{:d}*5 = {:d}'.format(7, 7*5))
print('{:d}*5 = {:d}'.format(8, 8*5))
print('{:d}*5 = {:d}'.format(9, 9*5))
print('{:d}*5 = {:d}'.format(10, 10*5))
```

When executed, the 10 results are printed quite nicely as

```
1*5 = 5
2*5 = 10
...
...
```

With a for loop, however, the very same printout may be produced by just two (!) lines of code:

```
for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: # Note... for, in and colon
print('{:d}*5 = {:d}'.format(i, i*5)) # Note indent
```

With this construction, the *loop variable* i takes on each of the values 1 to 10, and for each value, the print function is called.

The while Loop
The other basic loop construction in Python is the *while loop*, which runs as long as a condition is True. Let us move directly to an example, and explain what happens there, before we consider the loop more generally.

The Program In a program named ball_time.py, we may find the time of flight as the time when heights switch from positive to negative. The program could look like this

```
import numpy as np
v0 = 4.5 # Initial velocity
g = 9.81 # Acceleration of gravity
t = np.linspace(0, 1, 1000) # 1000 points in time interval
y = v0*t - 0.5*g*t**2 # Generate all heights
```

```
# Find index where ball approximately has reached y=0
i = 0
while y[i] >= 0:
i = i + 1
# Since y[i] is the height at time t[i], we do know the
# time as well when we have the index i...
print('Time of flight (in seconds): {:g}'.format(t[i]))
# We plot the path again just for comparison
import matplotlib.pyplot as plt
plt.plot(t, y)
plt.plot(t, 0*t, 'g--')
plt.xlabel('Time (s)')
plt.ylabel('Height (m)')
plt.show()
```

2. **Demonstrate Recursive Functions**

Definition of a recursive function: “A recursive function is a function defined in terms of itself via self-referential expressions.”
This means that the function will continue to call itself and repeat its behavior until some condition is met to return a result.

All recursive functions share a common structure made up of two parts: base case and recursive case.
To demonstrate this, let’s write a recursive function to calculate Factorial of n (n!):
Decompose the original problem into simpler instances of the same problem. This is the recursive case:

n! = n x (n−1) x (n−2) x (n−3) ···· x 3 x 2 x 1
n! = n x (n−1)!

As the large problem is broken down into successively less complex ones, those sub problems must eventually become so simple that they can be solved without further subdivision. This is the base case:

n! = n x (n−1)!
n! = n x (n−1) x (n−2)!
n! = n x (n−1) x (n−2) x (n−3)!
.
.
n! = n x (n−1) x (n−2) x (n−3) ···· x 3!
n! = n x (n−1) x (n−2) x (n−3) ···· x 3 x 2!
n! = n x (n−1) x (n−2) x (n−3) ···· x 3 x 2 x 1!

Here, 1! is our base case, and it equals 1.

Recursive function for calculating **n!** Implemented in Python:

```
def factorial_recursive(n):
   # Base case: 1! = 1
   if n == 1:
      return 1
   # Recursive case: n! = n * (n-1)!
```

```
    else:
        return n * factorial_recursive(n-1)
>>> factorial_recursive(5)
        120
```

3. Demonstrate Lists

How to create a list?

In Python programming, a list is created by placing all the items (elements) inside square brackets [], separated by commas.
It can have any number of items and they may be of different types (integer, float, string etc.).

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed data types
my_list = [1, "Hello", 3.4]
```

A list can also have another list as an item. This is called a nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Nested lists are accessed using nested indexing.

```
# List indexing
my_list = ['p', 'r', 'o', 'b', 'e']

# Output: p
print(my_list[0])
# Output: o
print(my_list[2])
# Output: e
print(my_list[4])
# Nested List
n_list = ["Happy", [2, 0, 1, 5]]
# Nested indexing
print(n_list[0][1])
print(n_list[1][3])
# Error! Only integer can be used for indexing
print(my_list[4.0])
```

Output

```
p
o
e
a
5
Traceback (most recent call last):
  File "<string>", line 21, in <module>
TypeError: list indices must be integers or slices, not float
```

Introduction to python Libraries

- Numpy
- Pandas
- Matplotlib
- Scikit

Essential Python Libraries For those who are less familiar with the scientific Python ecosystem and the libraries used throughout the Manual, introduction of each library given blow.

WEEK- 2:

SEQUENCE DATATYPES AND OBJECT-ORIENTED PROGRAMMING

Sequences, Mapping and Sets- Dictionaries- -Classes: Classes and Instances-Inheritance-

Exceptional Handling-Introduction to Regular Expressions using "re" module.

Lab Exercises

1. Demonstrate Tuples and Sets
2. Demonstrate Dictionaries
3. Demonstrate inheritance and exceptional handling
4. Demonstrate use of "re".

1.Write a program to Demonstrate Tuples and Sets

```
tuple1 = tuple(input("Enter the tuple elements ..."))

print(tuple1)

count = 0

for i in tuple1:
    print("tuple1[%d] = %s"%(count, i));
```

Output:

```
Enter the tuple elements ...12345

('1', '2', '3', '4', '5')

tuple1[0] = 1

tuple1[0] = 2

tuple1[0] = 3

tuple1[0] = 4

tuple1[0] = 5
```

OR

```
# empty tuple

my_tuple = ()

print(my_tuple)


# tuple having integers

my_tuple = (1, 2, 3)

print(my_tuple)


# tuple with mixed datatypes

my_tuple = (1, "Hello", 3.4)

print(my_tuple)


# nested tuple

my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

print(my_tuple)
```

```
# tuple can be created without parentheses

my_tuple = 3, 4.6, "dog"

print(my_tuple)


a, b, c = my_tuple

print(a)

print(b)

print(c)
```

Output:

```
()

(1, 2, 3)

(1, 'Hello', 3.4)

('mouse', [8, 4, 6], (1, 2, 3))

(3, 4.6, 'dog')

3

4.6

Dog
```

2. Demonstrate Dictionaries

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}

print(type(Employee))

print("printing Employee data .... ")

for x in Employee.items():

   print(x)


print("Enter the details of the new employee....");
```

```
Employee["Name"] = input("Name: ");
Employee["Age"] = int(input("Age: "));
Employee["salary"] = int(input("Salary: "));
Employee["Company"] = input("Company:");
print("printing the new data");
for x,y in Employee.items():
   print(x,y)
```

output:

```
<class 'dict'>
printing Employee data ....
('Name', 'John')
('Age', 29)
('salary', 25000)
('Company', 'GOOGLE')
Enter the details of the new employee....
Name: David
Age: 19
Salary: 8900
Company:Wipro
printing the new data
Name David
Age 19
Salary 8900
Company Wipro
```

Demonstrate inheritance and exceptional handling

Write a program to implement Inheritance

```
class Polygon:
  def __init__(self, no_of_sides):
    self.n = no_of_sides
    self.sides = [0 for i in range(no_of_sides)]

  def inputSides(self):
    self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in
range(self.n)]

  def dispSides(self):
    for i in range(self.n):
      print("Side",i+1,"is",self.sides[i])


class Triangle(Polygon):
  def __init__(self):
    Polygon.__init__(self,3)

  def findArea(self):
    a, b, c = self.sides
    # calculate the semi-perimeter
    s = (a + b + c) / 2
    area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
    print('The area of the triangle is %0.2f' %area)

t = Triangle()

t.inputSides()
t.dispSides()
t.findArea()
```

output:

```
Enter side 1 : 3
Enter side 2 : 5
Enter side 3 : 4

Side 1 is 3.0
Side 2 is 5.0
Side 3 is 4.0
The area of the triangle is 6.00
```

Write a program to Exception Handling.

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b;
    print("a/b = %d"%c)
except:
    print("can't divide by zero")
else:
    print("Hi I am else block")
```

output:

```
Enter a:10
Enter b:2
a/b = 5
Hi I am else block

Enter a:10
Enter b:0
can't divide by zero
```

Demonstrate use of “re”.

```
# Program to remove all whitespaces
import re
```

```
# multiline string

string = 'abc 12\

de 23 \n f45 6'

# matches all whitespace characters

pattern = '\s+'

# empty string

replace = ''

new_string = re.subn(pattern, replace, string)

print(new_string)
```

Output:

```
('abc12de23f456', 4)
```

WEEK-3

Lab Exercises
1. Demonstrate Aggregation
2. Demonstrate Indexing and Sorting

NumPy :
NumPy, short for Numerical Python, is the foundational package for scientific computing in Python. The majority of this book will be based on NumPy and libraries built on top of NumPy. It provides, among other things:
• A fast and efficient multidimensional array object *ndarray*
• Functions for performing element-wise computations with arrays or mathematical operations between arrays • Tools for reading and writing array-based data sets to disk
• Linear algebra operations, Fourier transform, and random number generation
• Tools for integrating connecting C, C++, and Fortran code to Python

Beyond the fast array-processing capabilities that NumPy adds to Python, one of its primary purposes with regards to data analysis is as the primary container for data to be passed between algorithms. For numerical data, NumPy arrays are a much more efficient way of storing and manipulating data than the other built-in Python data structures. Also, libraries written in a lower-level language, such as C or Fortran, can operate on the data stored in a NumPy array without copying any data.

1.Demonstrate Aggregation

Aggregation

- np.sum() - faster than .sum(), make demo, np is really fast
- np.mean()
- np.std()
- np.var()
- np.min()
- np.max()
- np.argmin() - find index of minimum value
- np.argmax() - find index of maximum value

```
sum(a1)
6
np.sum(a1)
6
```

Use NumPy's np.sum() on NumPy arrays and Python's sum() on Python lists.

```
massive_array = np.random.random(100000)
massive_array.size
100000
%timeit sum(massive_array) # Python sum()
%timeit np.sum(massive_array) # NumPy np.sum()
33 ms ± 2.08 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
97.3 µs ± 2.12 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
import random
massive_list = [random.randint(0, 10) for i in range(100000)]
len(massive_list)
100000
massive_list[:10]
[6, 3, 9, 1, 6, 0, 1, 1, 7, 3]
%timeit sum(massive_list)
%timeit np.sum(massive_list)
1.59 ms ± 70.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
11.3 ms ± 49.6 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
a2
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
# Find the mean
np.mean(a2)
3.6333333333333333
# Find the max
np.max(a2)
6.5
# Find the min
np.min(a2)
```

```
1.0
# Find the standard deviation
np.std(a2)
1.8226964152656422
# Find the variance
np.var(a2)
3.3222222222222224
# The standard deviation is the square root of the variance
np.sqrt(np.var(a2))
1.8226964152656422
```

2. Demonstrate Indexing and Sorting

```
import numpy as np

# 1-dimensonal array, also referred to as a vector

a1 = np.array([1, 2, 3])

# 2-dimensional array, also referred to as matrix

a2 = np.array([[1, 2.0, 3.3],  [4, 5, 6.5]])

# 3-dimensional array, also referred to as a matrix

a3 = np.array([[[1, 2, 3],  [4, 5, 6], [7, 8, 9]], [[10, 11, 12], [13, 14, 15], [16, 17, 18]]])

a1

array([1, 2, 3])
a2

array([[1. , 2. , 3.3],

       [4. , 5. , 6.5]])
```

```
a3

array([[[ 1,  2,  3],

        [ 4,  5,  6],

        [ 7,  8,  9]],

       [[10, 11, 12],

        [13, 14, 15],

        [16, 17, 18]]])

a1[0]

   1

a2[0]

array([1. , 2. , 3.3])
```

a3[0]

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

# Get 2nd row (index 1) of a2

a2[1]

```
array([4. , 5. , 6.5])
```

# Get the first 2 values of the first 2 rows of both arrays

a3[:2, :2, :2]

```
array([[[ 1,  2],
        [ 4,  5]],
       [[10, 11],
        [13, 14]]])
```

Sorting arrays

- **np.sort()**
- **np.argsort()**
- **np.argmax()**
- **np.argmin()**

random_array

```
array([[0, 0, 7],
       [3, 3, 2],
       [9, 4, 9],
       [4, 7, 2],
       [1, 9, 9]])
```

```
np.sort(random_array)
array([[0, 0, 7],
      [2, 3, 3],
      [4, 9, 9],
      [2, 4, 7],
      [1, 9, 9]])
np.argsort(random_array)
array([[0, 1, 2],
      [2, 0, 1],
      [1, 0, 2],
      [2, 0, 1],
      [0, 1, 2]], dtype=int64)
a1
array([1, 2, 3])
# Return the indices that would sort an array
np.argsort(a1)
array([0, 1, 2], dtype=int64)
# No axis
np.argmin(a1)
0
random_array
array([[0, 0, 7],
      [3, 3, 2],
      [9, 4, 9],
      [4, 7, 2],
      [1, 9, 9]])
# Down the vertical
np.argmax(random_array, axis=1)
array([2, 0, 0, 1, 1], dtype=int64)
# Across the horizontal
np.argmin(random_array, axis=0)
array([0, 0, 1], dtype=int64)
```

WEEK-4

Lab Exercises

1. Demonstrate handling of missing data
2. Demonstrate hierarchical indexing

Pandas:

pandas provides rich data structures and functions designed to make working with structured data fast, easy, and expressive. It is, as you will see, one of the critical ingredients enabling Python to be a powerful and productive data analysis environment. The primary object in pandas that will be used in this manual is the **DataFrame**, a two dimensional tabular, column-oriented data structure with both row and column labels:

```
>>> frame
```

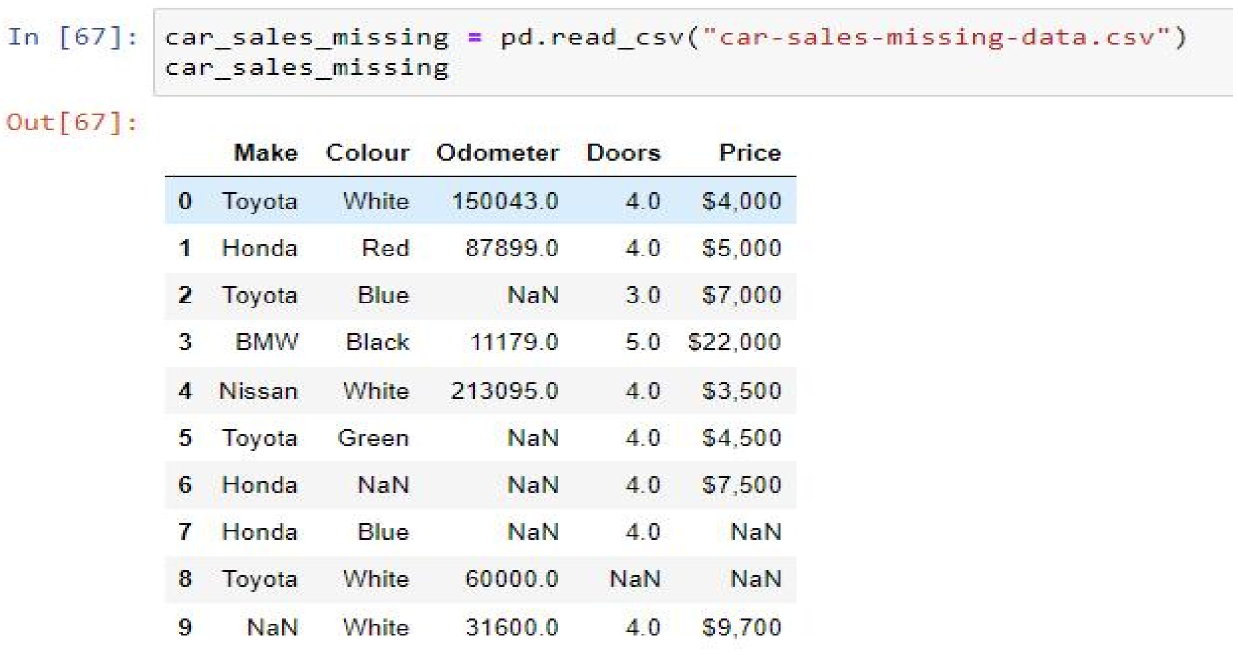
	total_bill	tip	sex	smoker	day	time	size
1	16.99	1.01	Female	No	Sun	Dinner	2
2	10.34	1.66	Male	No	Sun	Dinner	3
3	21.01	3.5	Male	No	Sun	Dinner	3
4	23.68	3.31	Male	No	Sun	Dinner	2
5	24.59	3.61	Female	No	Sun	Dinner	4
6	25.29	4.71	Male	No	Sun	Dinner	4
7	8.77	2	Male	No	Sun	Dinner	2
8	26.88	3.12	Male	No	Sun	Dinner	4
9	15.04	1.96	Male	No	Sun	Dinner	2
10	14.78	3.23	Male	No	Sun	Dinner	2

pandas combines the high performance array-computing features of NumPy with the flexible data manipulation capabilities of spreadsheets and relational databases (such as SQL). It provides sophisticated indexing functionality to make it easy to reshape, slice and dice, perform aggregations, and select subsets of data. pandas is the primary tool that we will use in this manual

For financial users, pandas feature rich, high-performance time series functionality and tools well-suited for working with financial data. In fact, I initially designed pandas as an ideal tool for financial data analysis applications.
For users of the R language for statistical computing, the **DataFrame** name will be familiar, as the object was named after the similar R **data.frame** object. They are not the same, however; the functionality provided by data.frame in R is essentially a strict subset of that provided by the pandas **DataFrame**. While this is a manual about Python, I will occasionally draw comparisons with R as it is one of the most widely-used open source data analysis environments and will be familiar to many readers.
The pandas name itself is derived from panel data, an econometrics term for multidimensional structured data sets, and Python data analysis itself.
1. Demonstrate handling of missing data

- Reassigning the column changes it in the original DataFrame. This trend occurs throughout all kinds of data manipulation with pandas.
- Some functions have a parameter called inplace which means a DataFrame is updated in place without having to reassign it.
- Let's see what it looks like in combination with .fillna(), a function which fills missing data. But the thing is, our table isn't missing any data.
- In practice, it's likely you'll work with datasets which aren't complete. What this means is you'll have to decide whether how to fill the missing data or remove the rows which have data missing.
- Let's check out what a version of our car_sales DataFrame might look like with missing values.

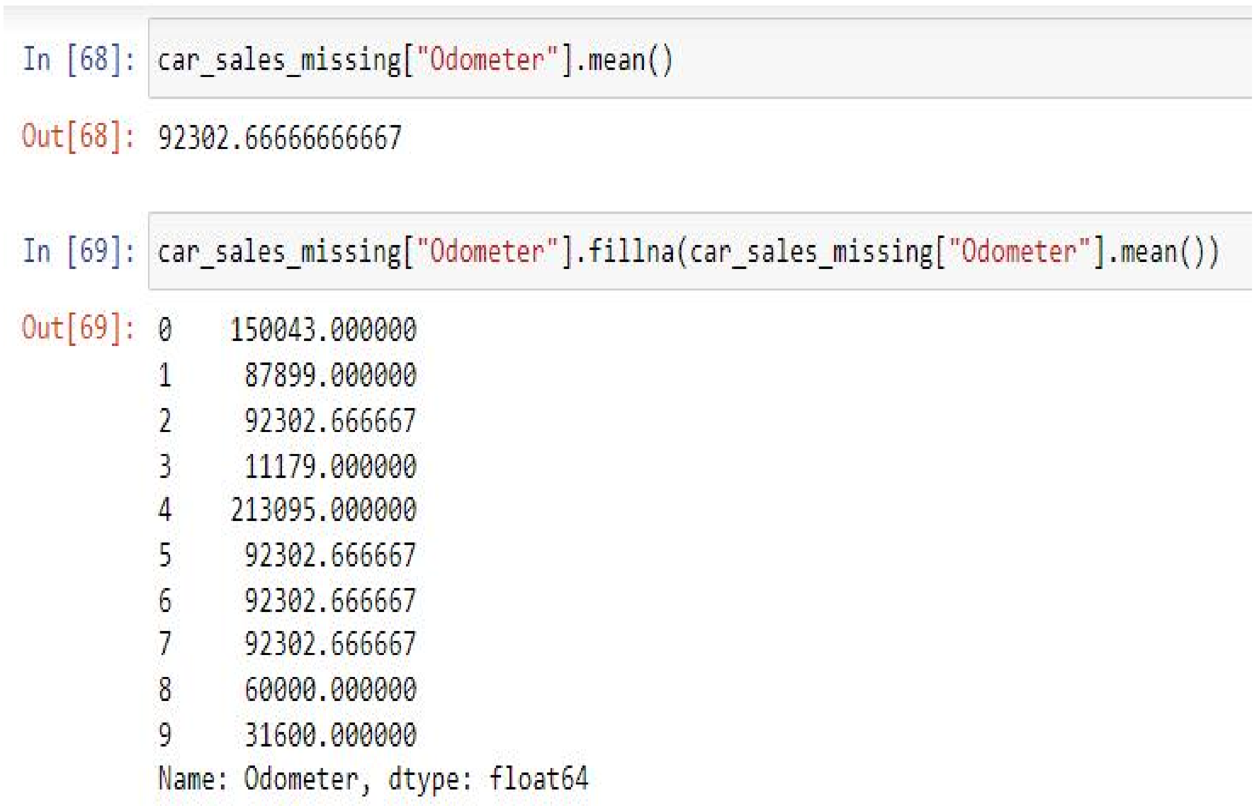
1. Demonstrate handling of missing data



In [67]:
```
car_sales_missing = pd.read_csv("car-sales-missing-data.csv")
car_sales_missing
```

Out[67]:

	Make	Colour	Odometer	Doors	Price
0	Toyota	White	150043.0	4.0	$4,000
1	Honda	Red	87899.0	4.0	$5,000
2	Toyota	Blue	NaN	3.0	$7,000
3	BMW	Black	11179.0	5.0	$22,000
4	Nissan	White	213095.0	4.0	$3,500
5	Toyota	Green	NaN	4.0	$4,500
6	Honda	NaN	NaN	4.0	$7,500
7	Honda	Blue	NaN	4.0	NaN
8	Toyota	White	60000.0	NaN	NaN
9	NaN	White	31600.0	4.0	$9,700

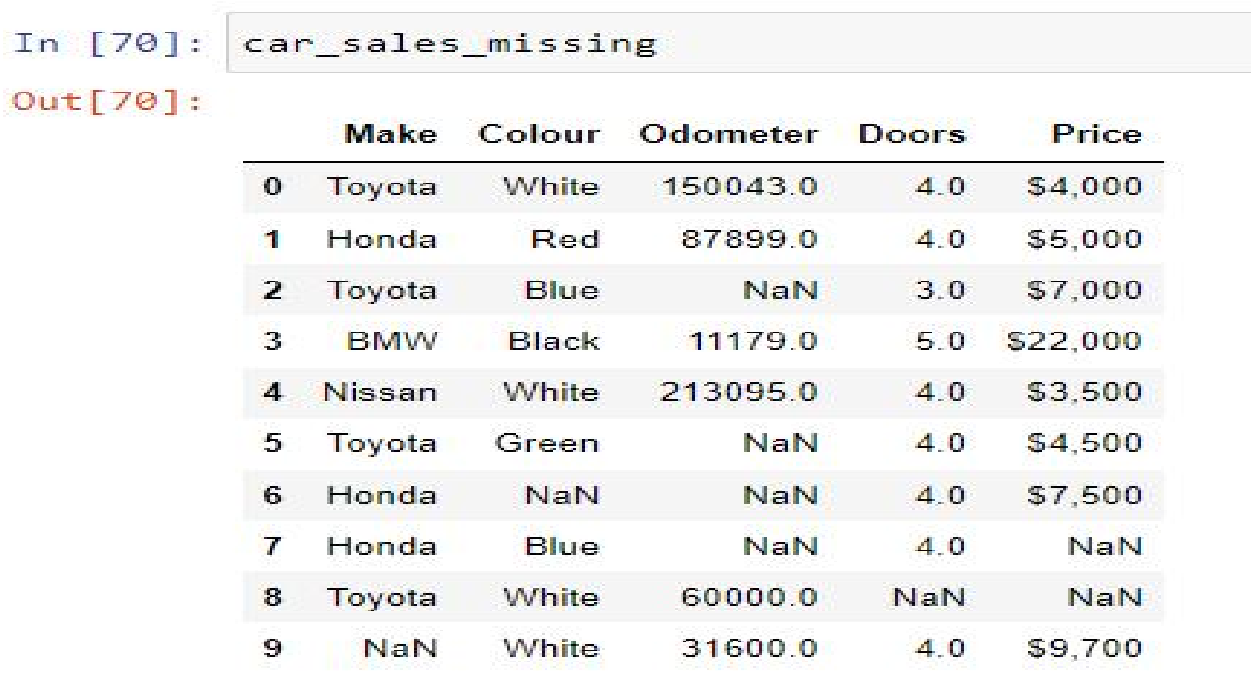


In [68]:
```
car_sales_missing["Odometer"].mean()
```

Out[68]:
```
92302.66666666667
```

In [69]:
```
car_sales_missing["Odometer"].fillna(car_sales_missing["Odometer"].mean())
```

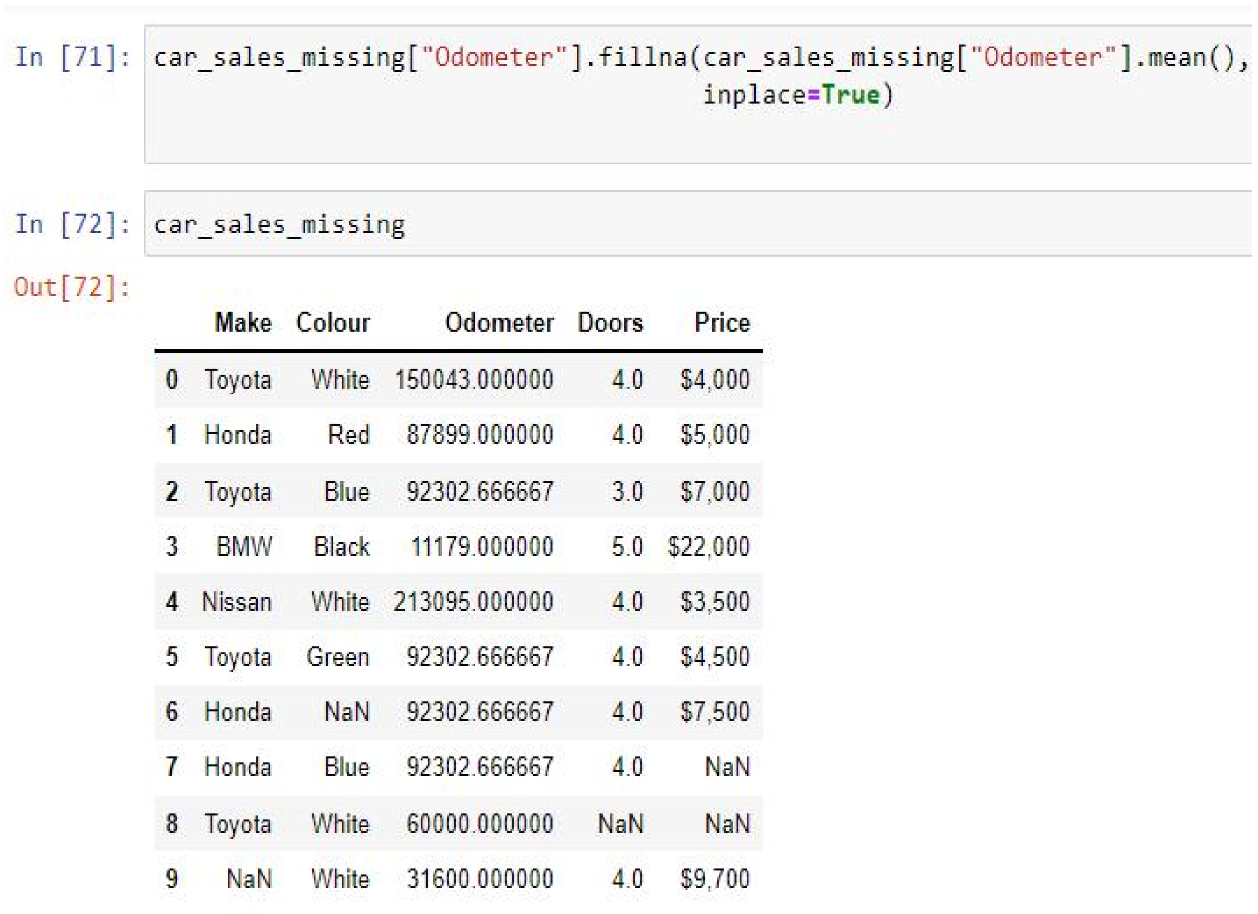
Out[69]:
```
0    150043.000000
1     87899.000000
2     92302.666667
3     11179.000000
4    213095.000000
5     92302.666667
6     92302.666667
7     92302.666667
8     60000.000000
9     31600.000000
Name: Odometer, dtype: float64
```



```
In [70]: car_sales_missing
```

Out[70]:

	Make	Colour	Odometer	Doors	Price
0	Toyota	White	150043.0	4.0	$4,000
1	Honda	Red	87899.0	4.0	$5,000
2	Toyota	Blue	NaN	3.0	$7,000
3	BMW	Black	11179.0	5.0	$22,000
4	Nissan	White	213095.0	4.0	$3,500
5	Toyota	Green	NaN	4.0	$4,500
6	Honda	NaN	NaN	4.0	$7,500
7	Honda	Blue	NaN	4.0	NaN
8	Toyota	White	60000.0	NaN	NaN
9	NaN	White	31600.0	4.0	$9,700

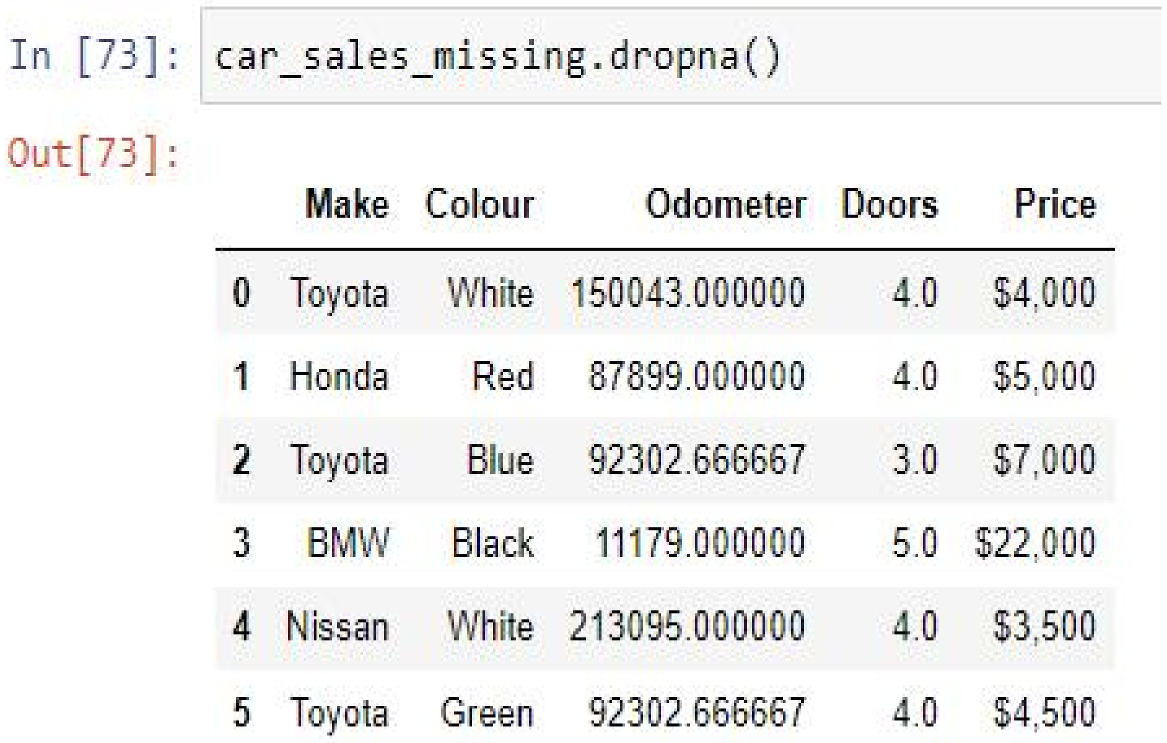


```
In [71]: car_sales_missing["Odometer"].fillna(car_sales_missing["Odometer"].mean(),
                                                inplace=True)
```

```
In [72]: car_sales_missing
```

Out[72]:

	Make	Colour	Odometer	Doors	Price
0	Toyota	White	150043.000000	4.0	$4,000
1	Honda	Red	87899.000000	4.0	$5,000
2	Toyota	Blue	92302.666667	3.0	$7,000
3	BMW	Black	11179.000000	5.0	$22,000
4	Nissan	White	213095.000000	4.0	$3,500
5	Toyota	Green	92302.666667	4.0	$4,500
6	Honda	NaN	92302.666667	4.0	$7,500
7	Honda	Blue	92302.666667	4.0	NaN
8	Toyota	White	60000.000000	NaN	NaN
9	NaN	White	31600.000000	4.0	$9,700



```
In [73]: car_sales_missing.dropna()

Out[73]:
```

	Make	Colour	Odometer	Doors	Price
0	Toyota	White	150043.000000	4.0	$4,000
1	Honda	Red	87899.000000	4.0	$5,000
2	Toyota	Blue	92302.666667	3.0	$7,000
3	BMW	Black	11179.000000	5.0	$22,000
4	Nissan	White	213095.000000	4.0	$3,500
5	Toyota	Green	92302.666667	4.0	$4,500

WEEK-5

1. Demonstrate usage of Pivot table
2. Demonstrate use of and query ()

```
import pandas as pd

# 2 main datatypes

series = pd.Series(["BMW", "Toyota", "Honda","BMW"])

series
```

OUT:

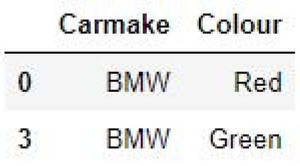
```
0       BMW
1    Toyota
2     Honda
3       BMW
dtype: object
```

```
colours = pd.Series(["Red", "Blue", "White","Green"])

colours
```

```
0      Red
1     Blue
2    White
3    Green
dtype: object
```

```
# DataFrame = 2-dimensional

car_data = pd.DataFrame({"Carmake": series, "Colour": colours})

car_data

car_data.query("(Carmake=='BMW') or (Colour=='Green')")
```

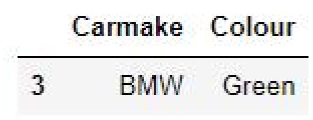
out:



	Carmake	Colour
0	BMW	Red
3	BMW	Green

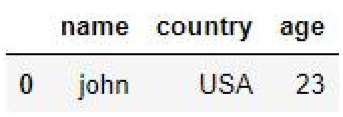
2. Demonstrate use of and query ()

```
car_data.query("(Carmake=='BMW') and (Colour=='Green')")
```



```
   Carmake  Colour
3      BMW   Green
```

```
df = pd.DataFrame({

    'name':['john','david','anna'],

    'country':['USA','UK', 'USA'],

    'age':[23,45,45]

})


df.query("(country=='USA') and (age==23)")
```

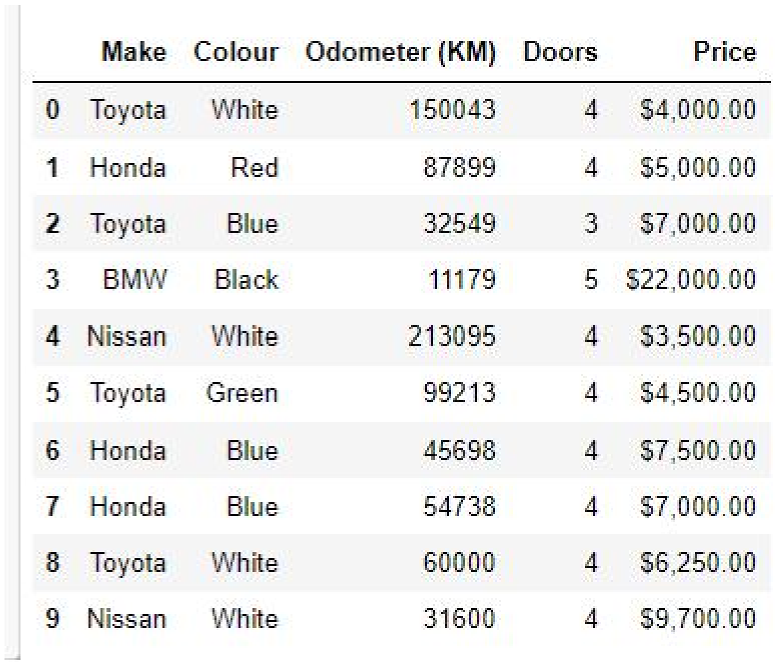


```
   name  country  age
0  john      USA   23
```

---

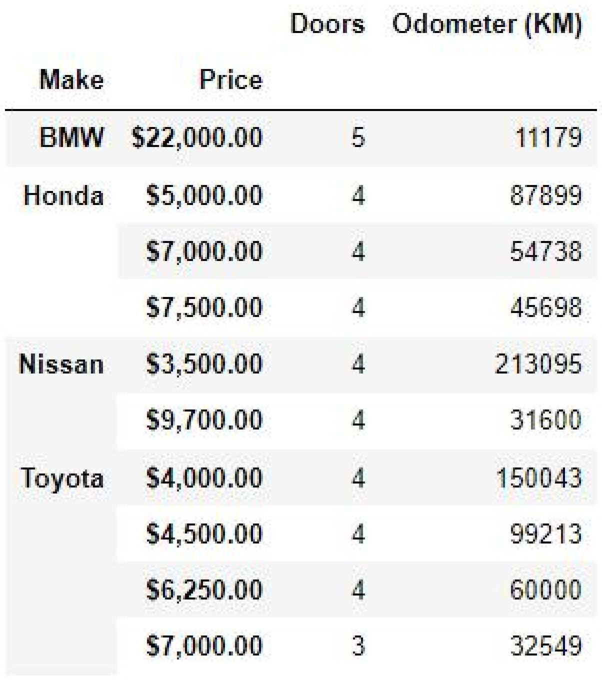
1. Demonstrate usage of Pivot table

```
#PIVOT  TABLE

# Import data

car_sales = pd.read_csv("car-sales.csv")

car_sales
```



	Make	Colour	Odometer (KM)	Doors	Price
0	Toyota	White	150043	4	$4,000.00
1	Honda	Red	87899	4	$5,000.00
2	Toyota	Blue	32549	3	$7,000.00
3	BMW	Black	11179	5	$22,000.00
4	Nissan	White	213095	4	$3,500.00
5	Toyota	Green	99213	4	$4,500.00
6	Honda	Blue	45698	4	$7,500.00
7	Honda	Blue	54738	4	$7,000.00
8	Toyota	White	60000	4	$6,250.00
9	Nissan	White	31600	4	$9,700.00

```
table = pd.pivot_table(car_sales,index=['Make','Price'])
table
```



		Doors	Odometer (KM)
Make	**Price**		
BMW	**$22,000.00**	5	11179
Honda	**$5,000.00**	4	87899
	$7,000.00	4	54738
	$7,500.00	4	45698
Nissan	**$3,500.00**	4	213095
	$9,700.00	4	31600
Toyota	**$4,000.00**	4	150043
	$4,500.00	4	99213
	$6,250.00	4	60000
	$7,000.00	3	32549

WEEK-6

1. Demonstrate Scatter Plot
2. Demonstrate 3D plotting

matplotlib
matplotlib is the most popular Python library for producing plots and other 2D data visualizations. It was originally created by John D. Hunter (JDH) and is now maintained by a large team of developers. It is well-suited for creating plots suitable for publication. It integrates well with IPython (see below), thus providing a comfortable interactive environment for plotting and exploring data. The plots are also interactive; you can zoom in on a section of the plot and pan around the plot using the toolbar in the plot window.

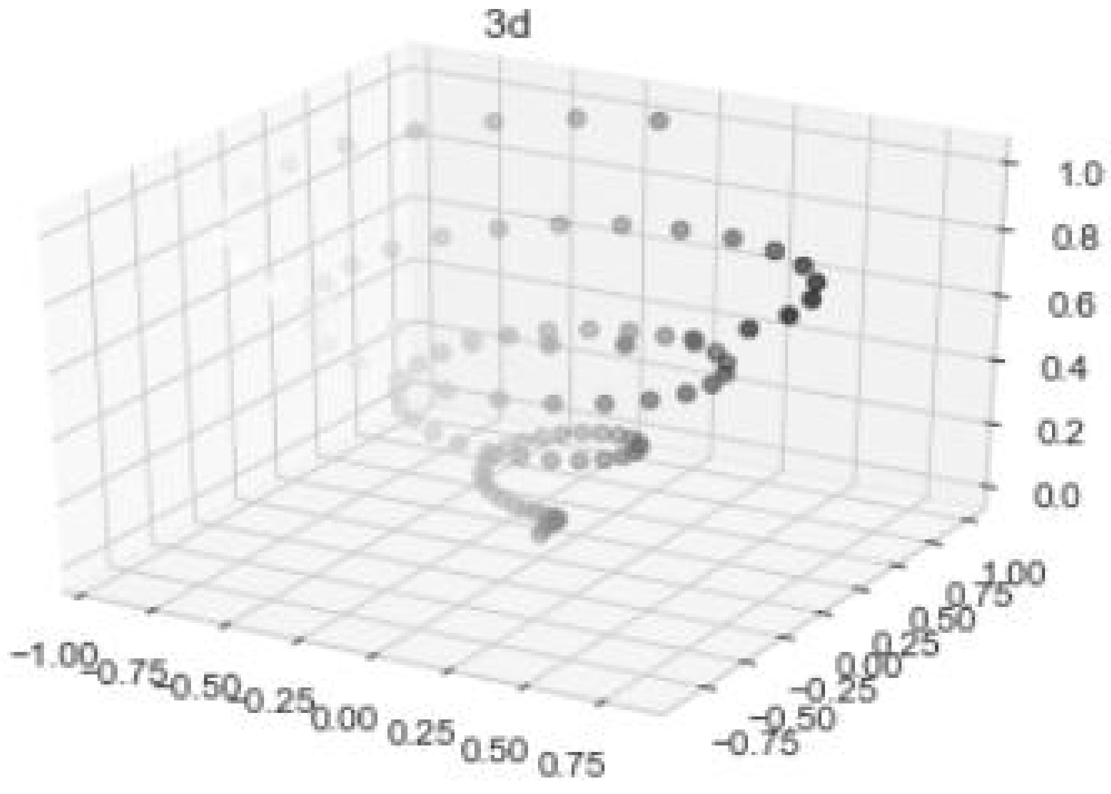
Import Conventions
The Python community has adopted a number of naming conventions for commonly used modules:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

This means that when you see np.arange, this is a reference to the arange function in NumPy. This is done as it’s considered bad practice in Python software development to import everything *(from numpy import *)* from a large package like NumPy.

2. Demonstrate 3D plotting

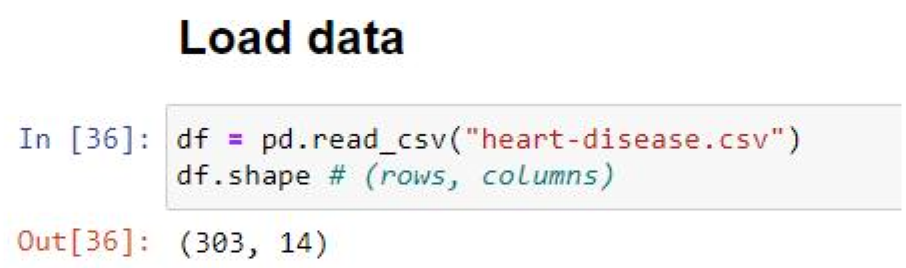
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure()

# syntax for 3-D projection
ax = plt.axes(projection ='3d')

# defining axes
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
c = x + y
ax.scatter(x, y, z, c = c)

# syntax for plotting
ax.set_title('3d ')
plt.show()
```


3d
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

WEEK-7

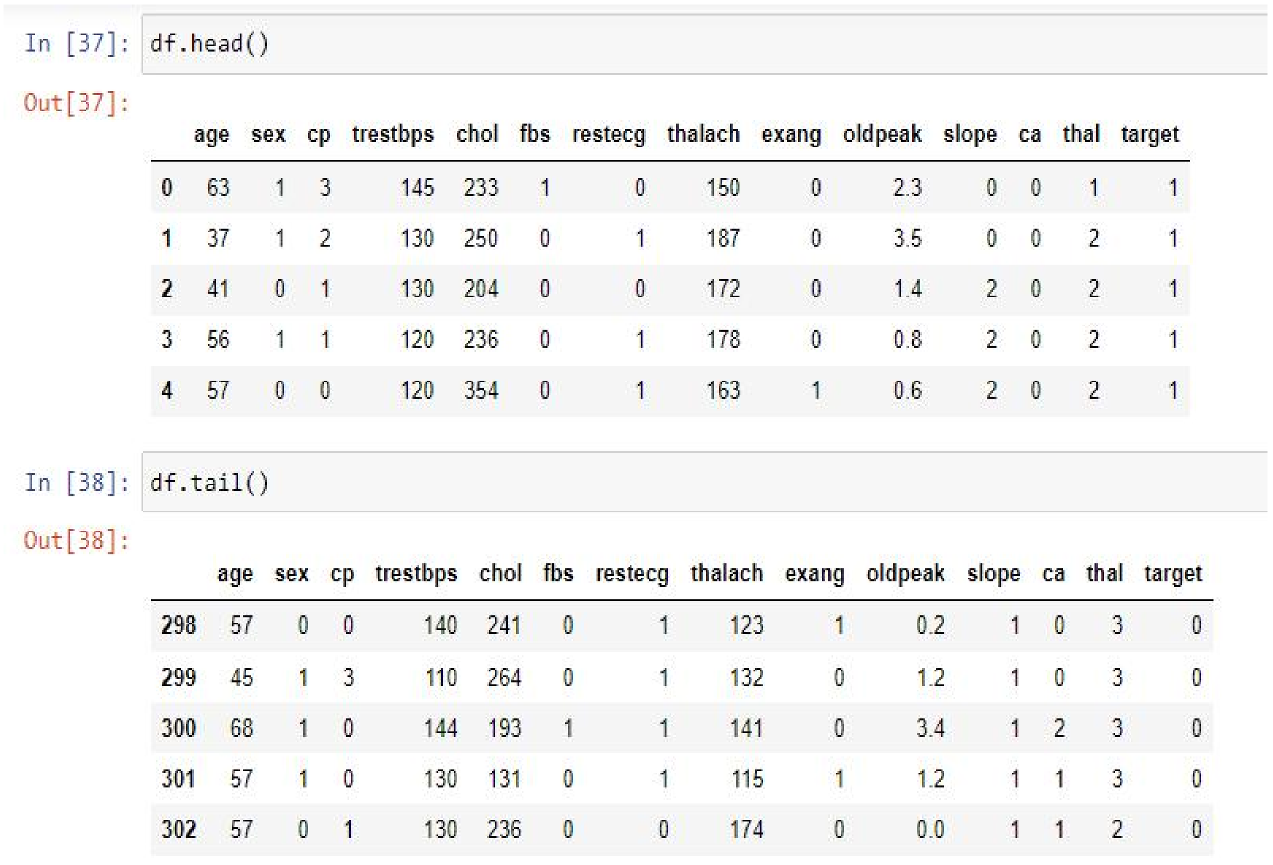
Week 7: Perform Data exploration and pre-processing in Python

Perform Data exploration and pre-processing in Python



Load data

```
In [36]: df = pd.read_csv("heart-disease.csv")
         df.shape # (rows, columns)

Out[36]: (303, 14)
```



```
In [37]: df.head()

Out[37]:
```

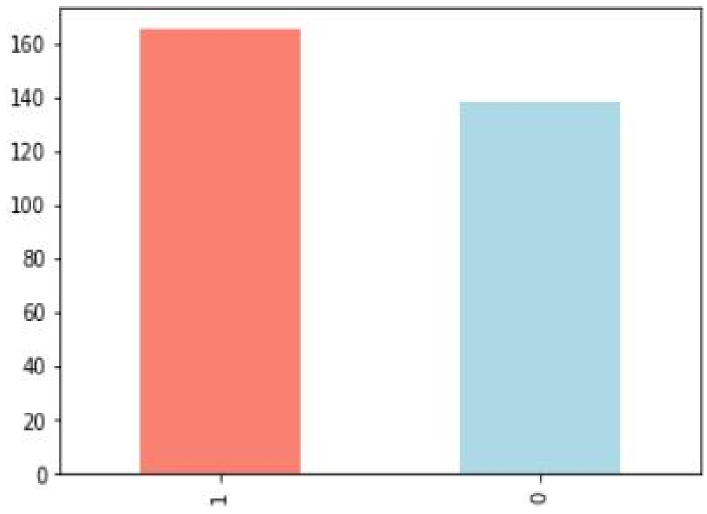
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
In [38]: df.tail()

Out[38]:
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

```
In [39]: # Let's find out how many of each class there
         df["target"].value_counts()

Out[39]: 1    165
         0    138
         Name: target, dtype: int64
```

```
In [40]: df["target"].value_counts().plot(kind="bar", color=["salmon", "lightblue"]);
```



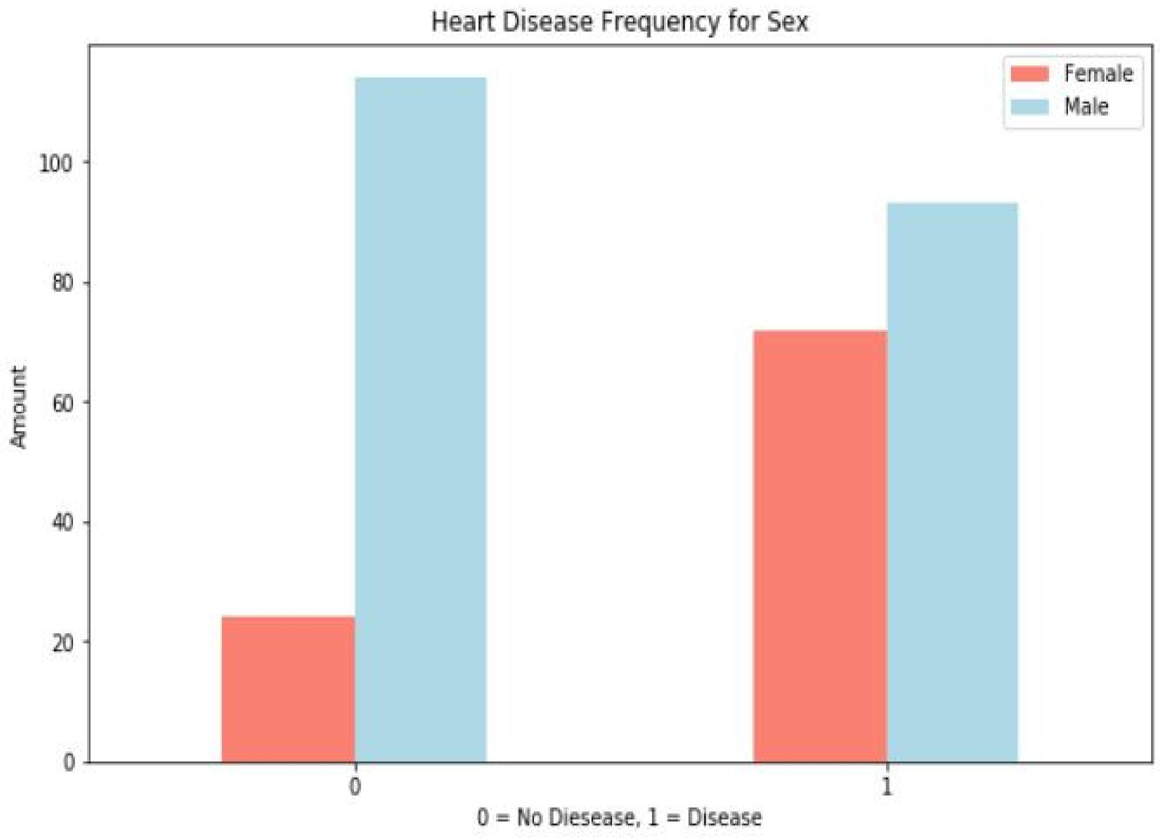
Heart Disease Frequency according to Sex

```
In [44]: df.sex.value_counts()

Out[44]: 1    207
         0     96
         Name: sex, dtype: int64
```

```
In [45]: # Compare target column with sex column
         df.sex.value_counts()
         pd.crosstab(df.target, df.sex)

Out[45]:
```

sex	0	1
target		
0	24	114
1	72	93

```
# Create a plot of crosstab
pd.crosstab(df.target, df.sex).plot(kind="bar",
                                    figsize=(10, 6),
                                    color=["salmon", "lightblue"])

plt.title("Heart Disease Frequency for Sex")
plt.xlabel("0 = No Diesease, 1 = Disease")
plt.ylabel("Amount")
plt.legend(["Female", "Male"]);
plt.xticks(rotation=0);
```



Week 8: Implement regularised linear regression

Week 9: Implement Naive Bayes classifier for dataset stored as CSV file.

Week 10: Implement regularized logistic regression

Week 11: Build models using different Ensembling techniques

Week 12: Build models using Decision trees

Week 13: Build model using SVM with different kernels

Week 14: Implement K-NN algorithm to classify a dataset.

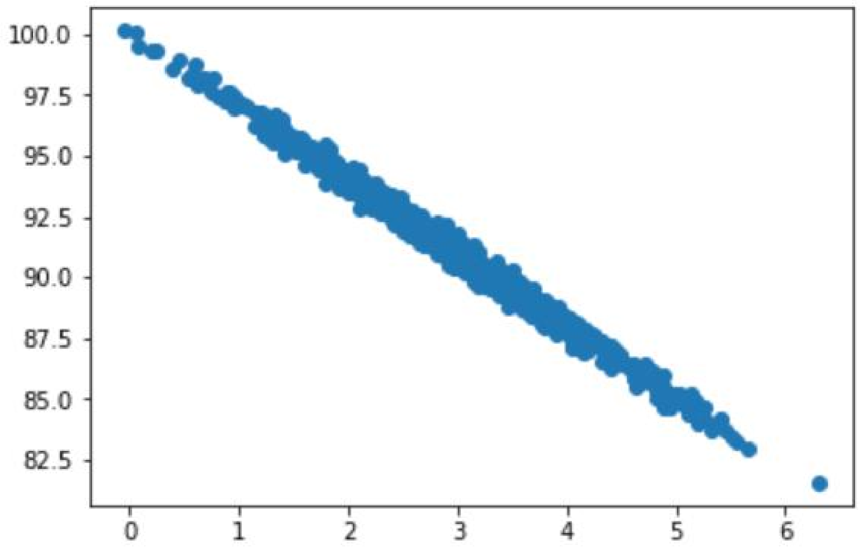
Week 15: Build model to perform Clustering using K-means after applying PCA and determining the value of K using Elbow method.

Implement regularised linear regression

Linear Regression

```
%matplotlib inline
import numpy as np
from pylab import *

pageSpeeds = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = 100 - (pageSpeeds + np.random.normal(0, 0.1, 1000)) * 3

scatter(pageSpeeds, purchaseAmount)
```

```
<matplotlib.collections.PathCollection at 0x1c90705bb38>
```

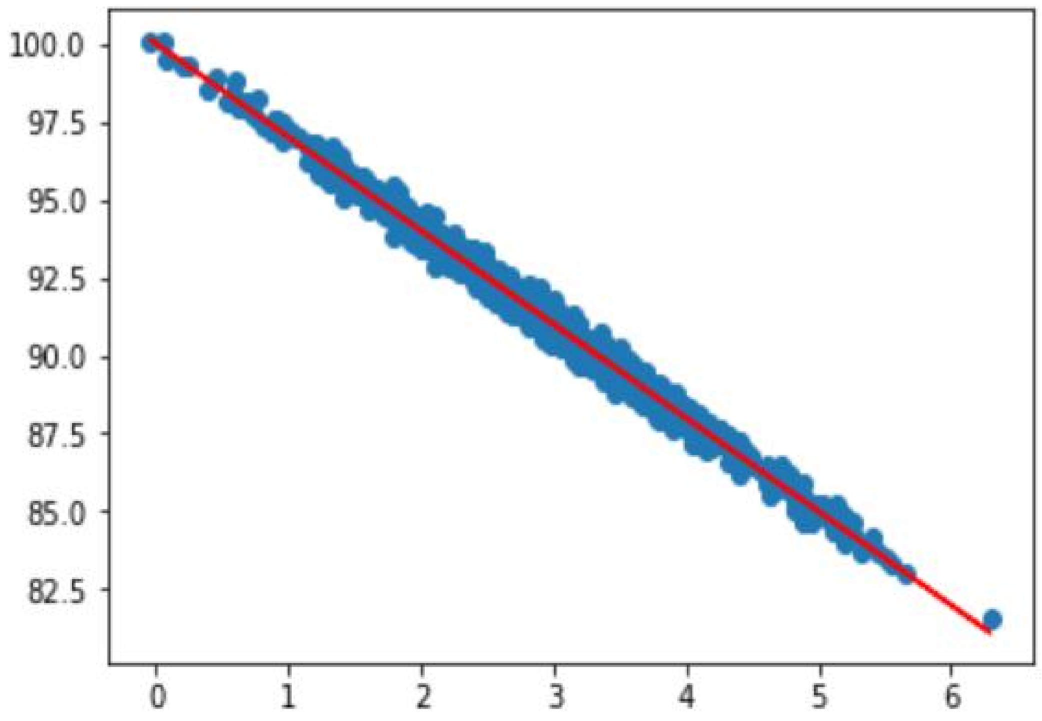


```
from scipy import stats

slope, intercept, r_value, p_value, std_err = stats.linregress(pageSpeeds, purchaseAmount)
```

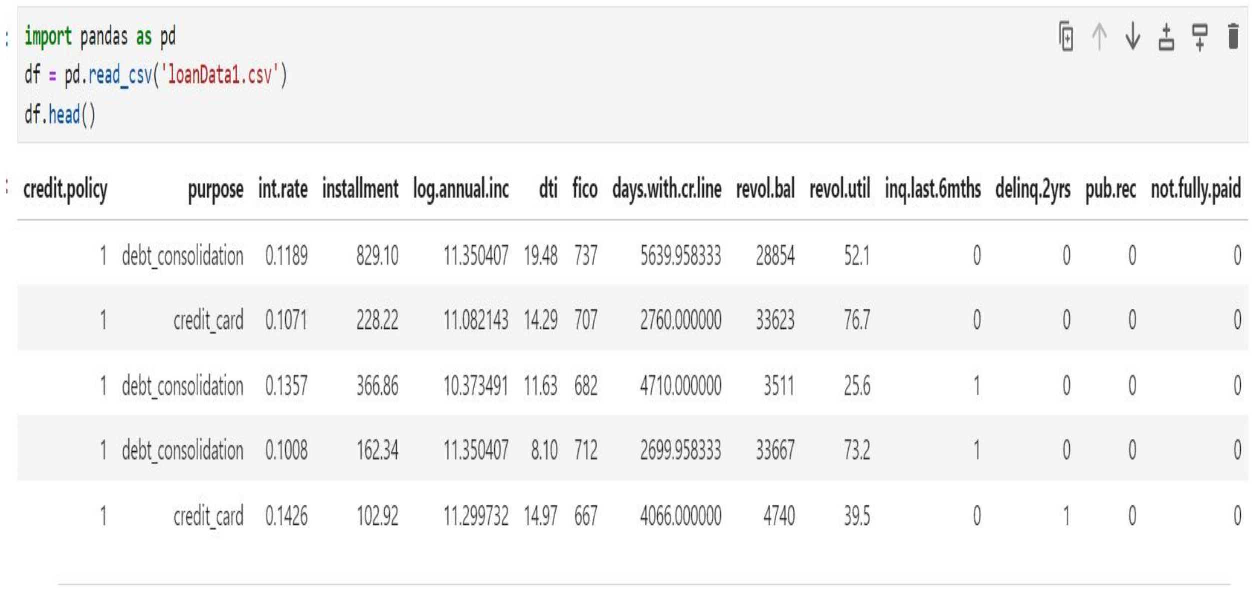
```
r_value ** 2
```

```
0.990903801365279
```

```python
import matplotlib.pyplot as plt

def predict(x):
    return slope * x + intercept

fitLine = predict(pageSpeeds)

plt.scatter(pageSpeeds, purchaseAmount)
plt.plot(pageSpeeds, fitLine, c='r')
plt.show()
```



Week 9: Implement Naive Bayes classifier for dataset stored as CSV file.



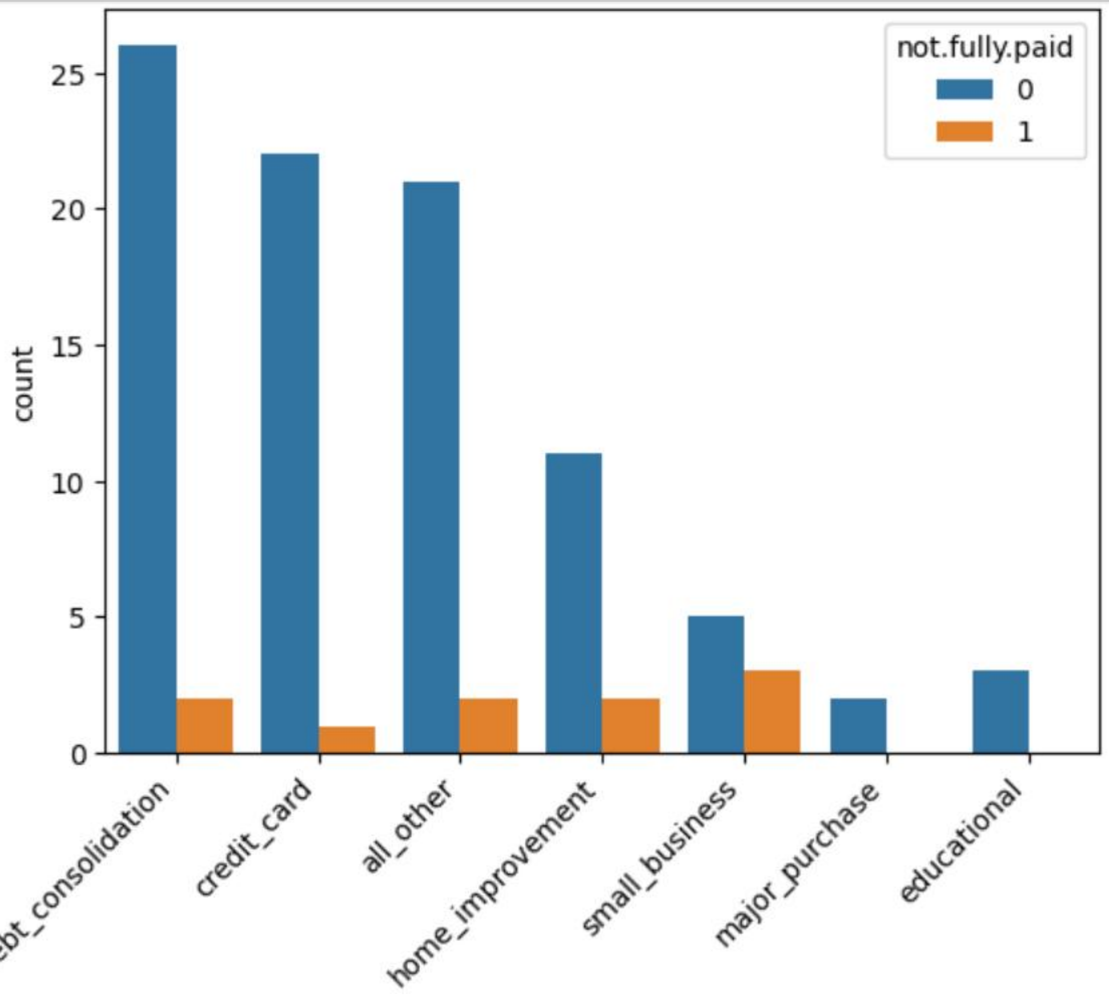
```
import pandas as pd
df = pd.read_csv('loanData1.csv')
df.head()
```

credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      100 non-null    int64  
 1   purpose            100 non-null    object 
 2   int.rate           100 non-null    float64
 3   installment        100 non-null    float64
 4   log.annual.inc     100 non-null    float64
 5   dti                100 non-null    float64
 6   fico               100 non-null    int64  
 7   days.with.cr.line  100 non-null    float64
 8   revol.bal          100 non-null    int64  
 9   revol.util         100 non-null    float64
 10  inq.last.6mths     100 non-null    int64  
 11  delinq.2yrs        100 non-null    int64  
 12  pub.rec            100 non-null    int64  
 13  not.fully.paid     100 non-null    int64  
dtypes: float64(6), int64(7), object(1)
memory usage: 11.1+ KB
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot(data=df,x='purpose',hue='not.fully.paid')
plt.xticks(rotation=45, ha='right');
```

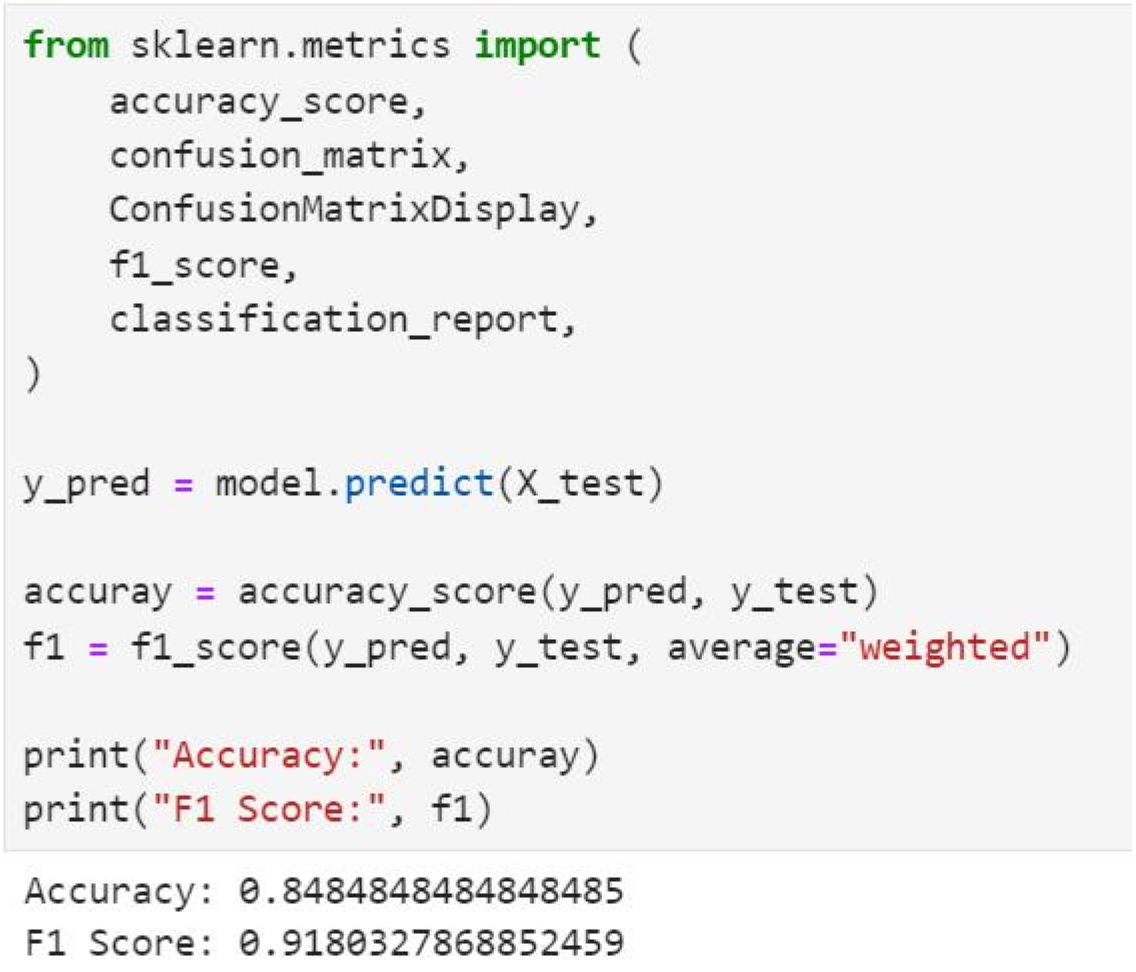


```
pre_df = pd.get_dummies(df,columns=['purpose'],drop_first=True)
pre_df.head()
```

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid	purpose_credit_
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0	
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0	
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0	
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0	
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0	

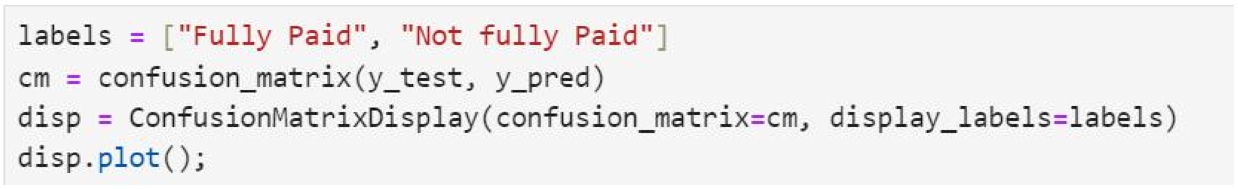
```python
from sklearn.model_selection import train_test_split

X = pre_df.drop('not.fully.paid', axis=1)
y = pre_df['not.fully.paid']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=125
)
```

```python
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()

model.fit(X_train, y_train);
```

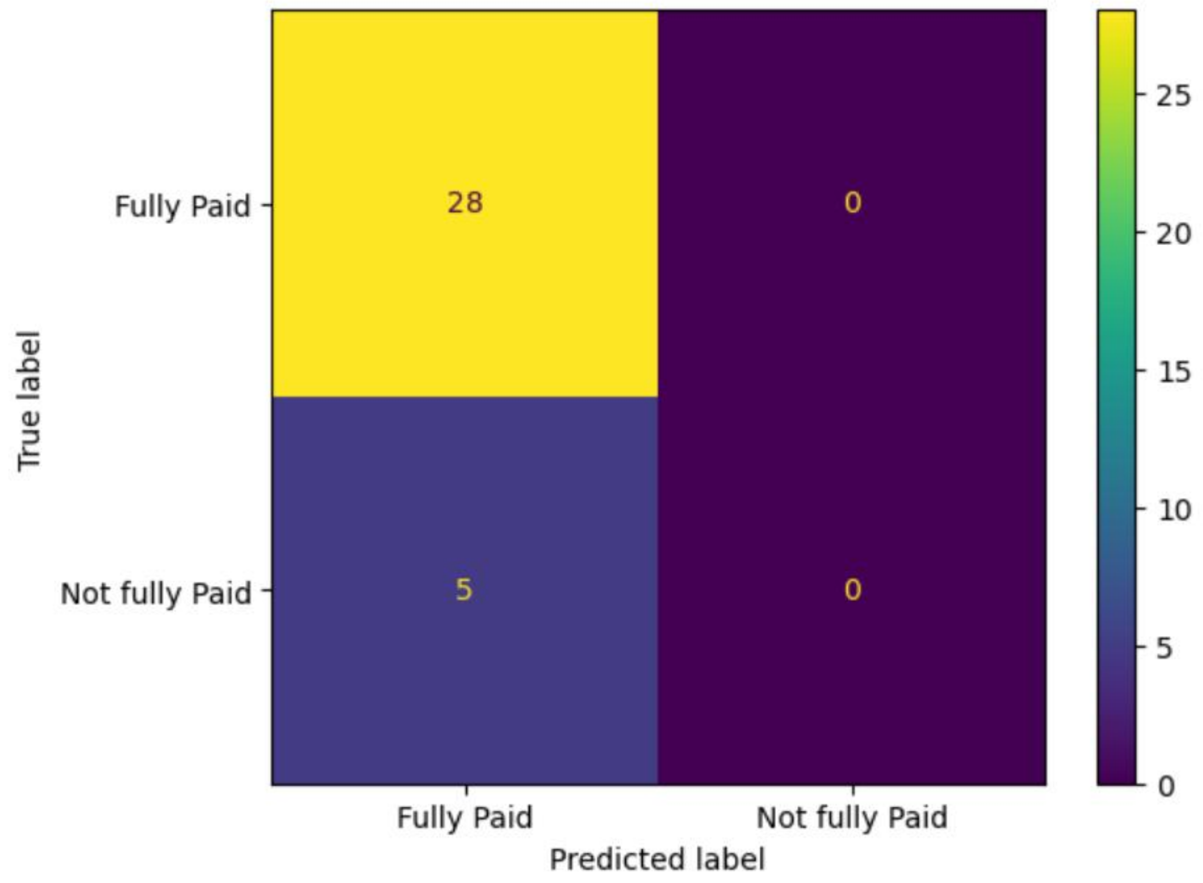


```python
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    ConfusionMatrixDisplay,
    f1_score,
    classification_report,
)

y_pred = model.predict(X_test)

accuray = accuracy_score(y_pred, y_test)
f1 = f1_score(y_pred, y_test, average="weighted")

print("Accuracy:", accuray)
print("F1 Score:", f1)
```

```
Accuracy: 0.8484848484848485
F1 Score: 0.9180327868852459
```



```python
labels = ["Fully Paid", "Not fully Paid"]
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot();
```



Week 10: Implement regularized logistic regression

Logistic regression

```python
import numpy as np
import pandas as pd
```

```python
data = pd.read_csv('Social_Network_Ads.csv')
```

```python
data.head()
```

```
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

```
data.Purchased.unique() #Binary classification
```

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

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

```
#Seperate your data as features and label
# Rule for Classification in SKlearn
# Expects your features to be in 2 dimension
# Expects your label to be in 1 dimension

features = data.iloc[:,[2,3]].values
label = data.iloc[:,4].values
```

```
#Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,
                                                 label,
                                                 test_size=0.2,
                                                 random_state = 10)
```

```
# Train the model
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

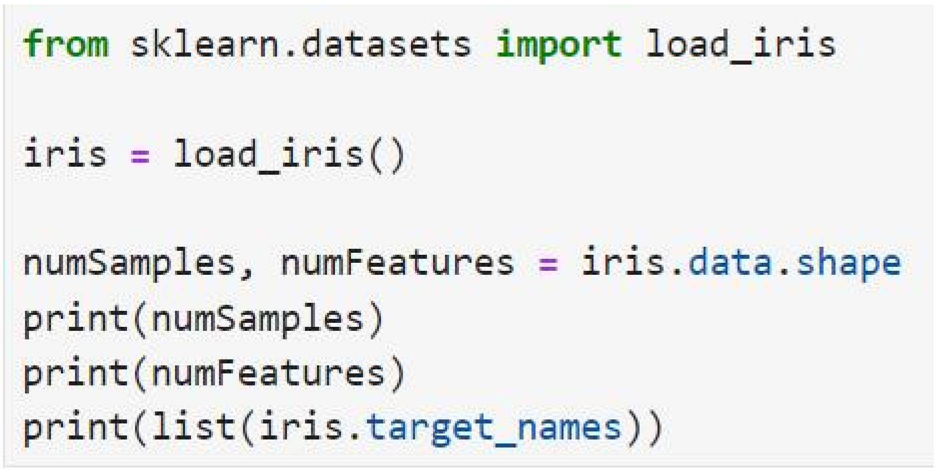
```
LogisticRegression()
```

```
#Check for Generalization
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

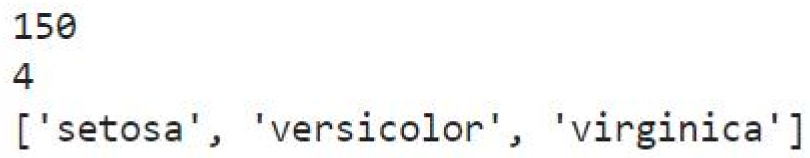
```
0.640625
0.65
```

Week 11: Build models using different Ensembling techniques

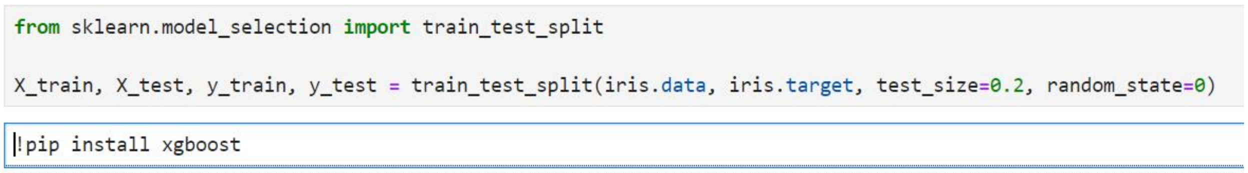
Ensembling Techniques:XGBoost



```python
from sklearn.datasets import load_iris

iris = load_iris()

numSamples, numFeatures = iris.data.shape
print(numSamples)
print(numFeatures)
print(list(iris.target_names))
```

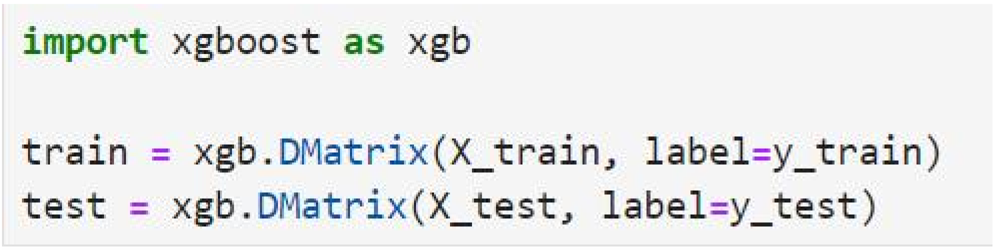


```
150
4
['setosa', 'versicolor', 'virginica']
```

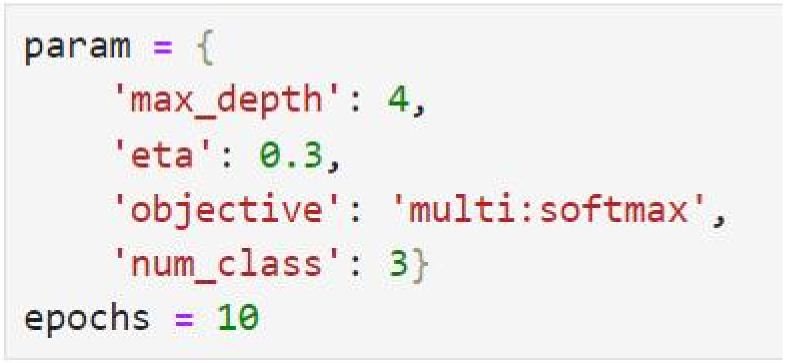


```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=0)
```

```
!pip install xgboost
```



```python
import xgboost as xgb

train = xgb.DMatrix(X_train, label=y_train)
test = xgb.DMatrix(X_test, label=y_test)
```



```python
param = {
    'max_depth': 4,
    'eta': 0.3,
    'objective': 'multi:softmax',
    'num_class': 3}
epochs = 10
```

```
model = xgb.train(param, train, epochs)
```

```
predictions = model.predict(test)
```

```
print(predictions)
```

```
[2. 1. 0. 2. 0. 2. 0. 1. 1. 1. 2. 1. 1. 1. 1. 0. 1. 1. 0. 0. 2. 1. 0. 0.
 2. 0. 0. 1. 1. 0.]
```

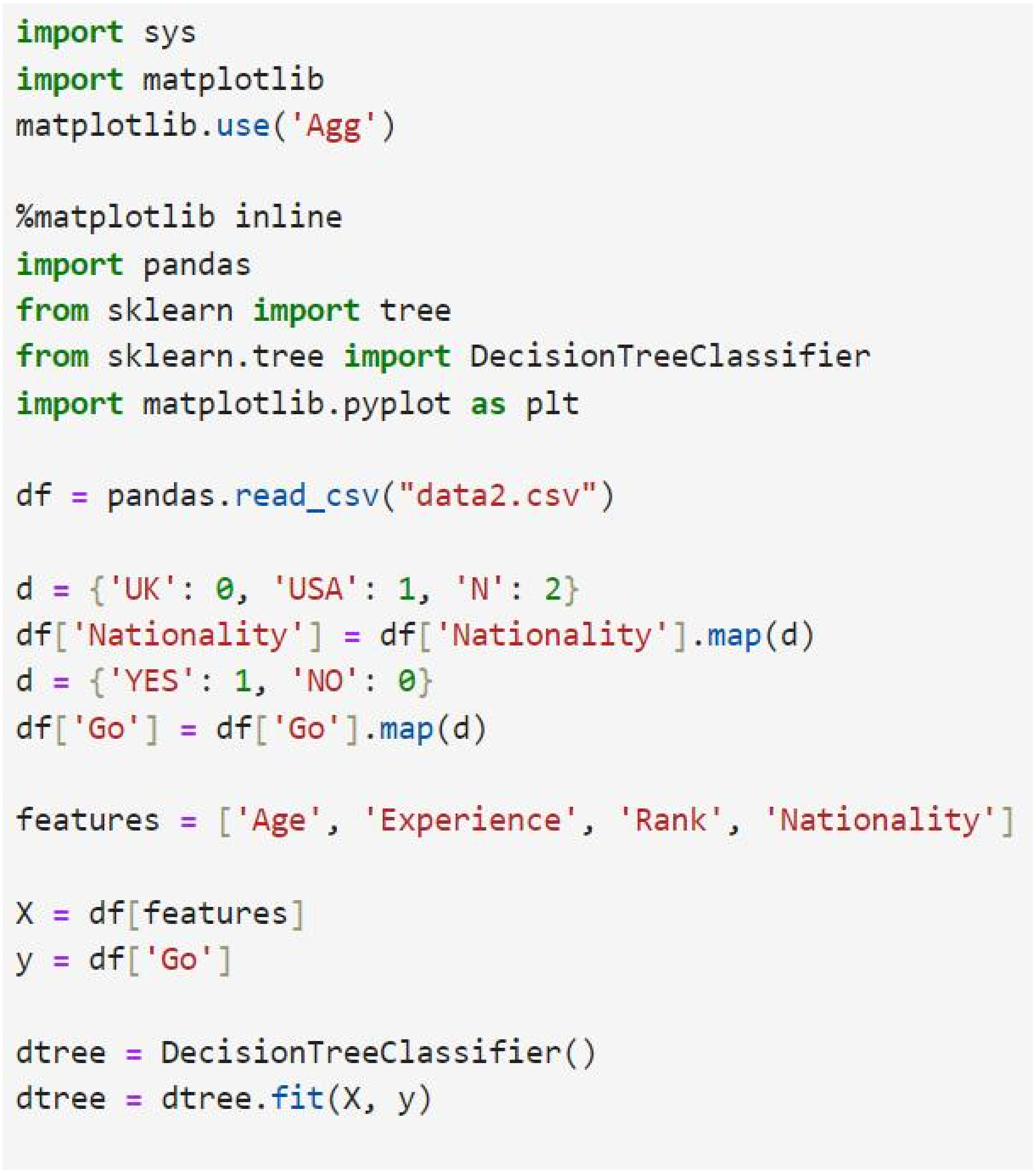
Let's measure the accuracy on the test data...

```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, predictions)
```

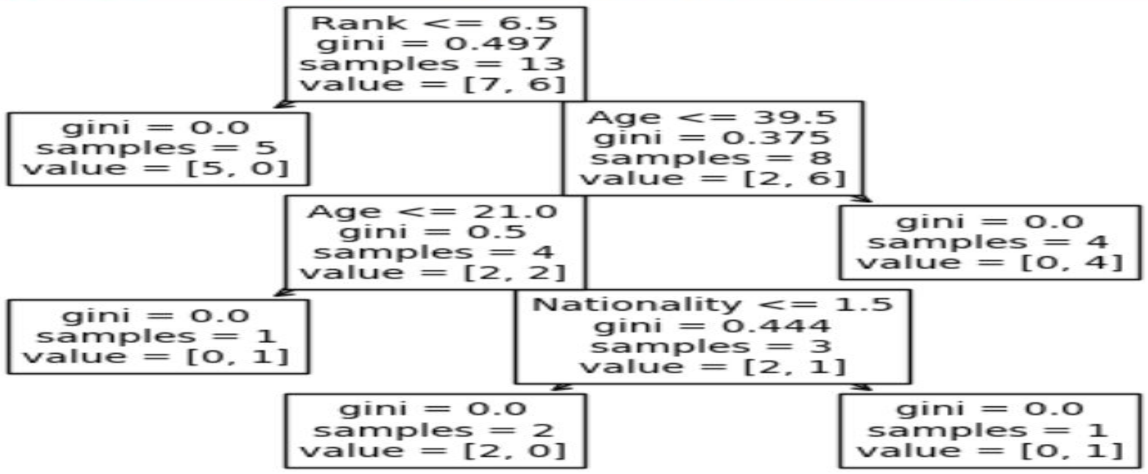
```
1.0
```

Week 12: Build models using Decision trees

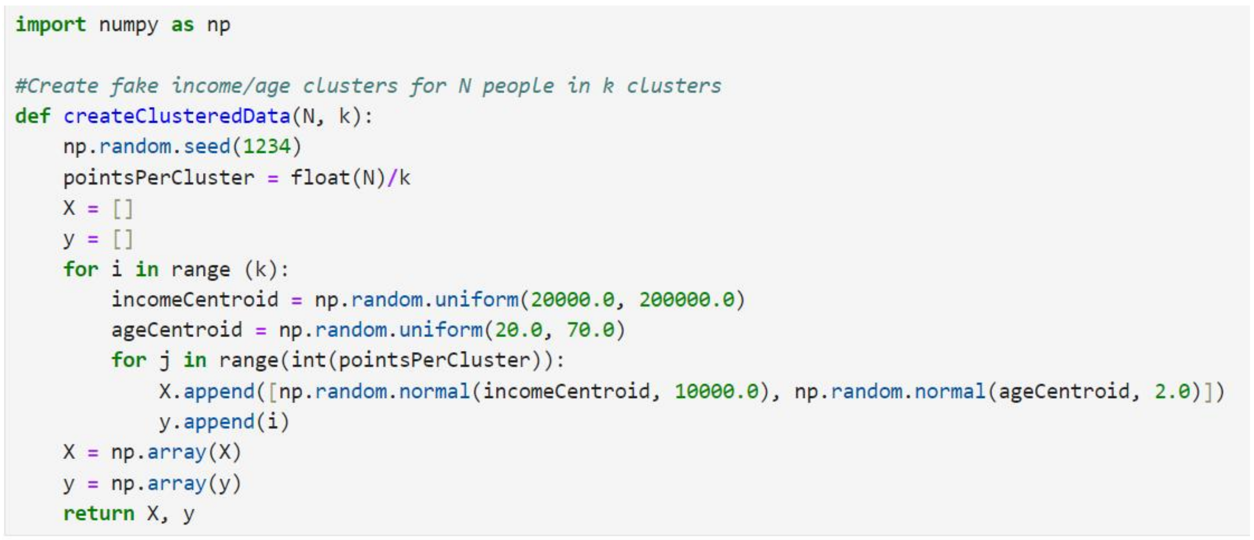
Decision Tree



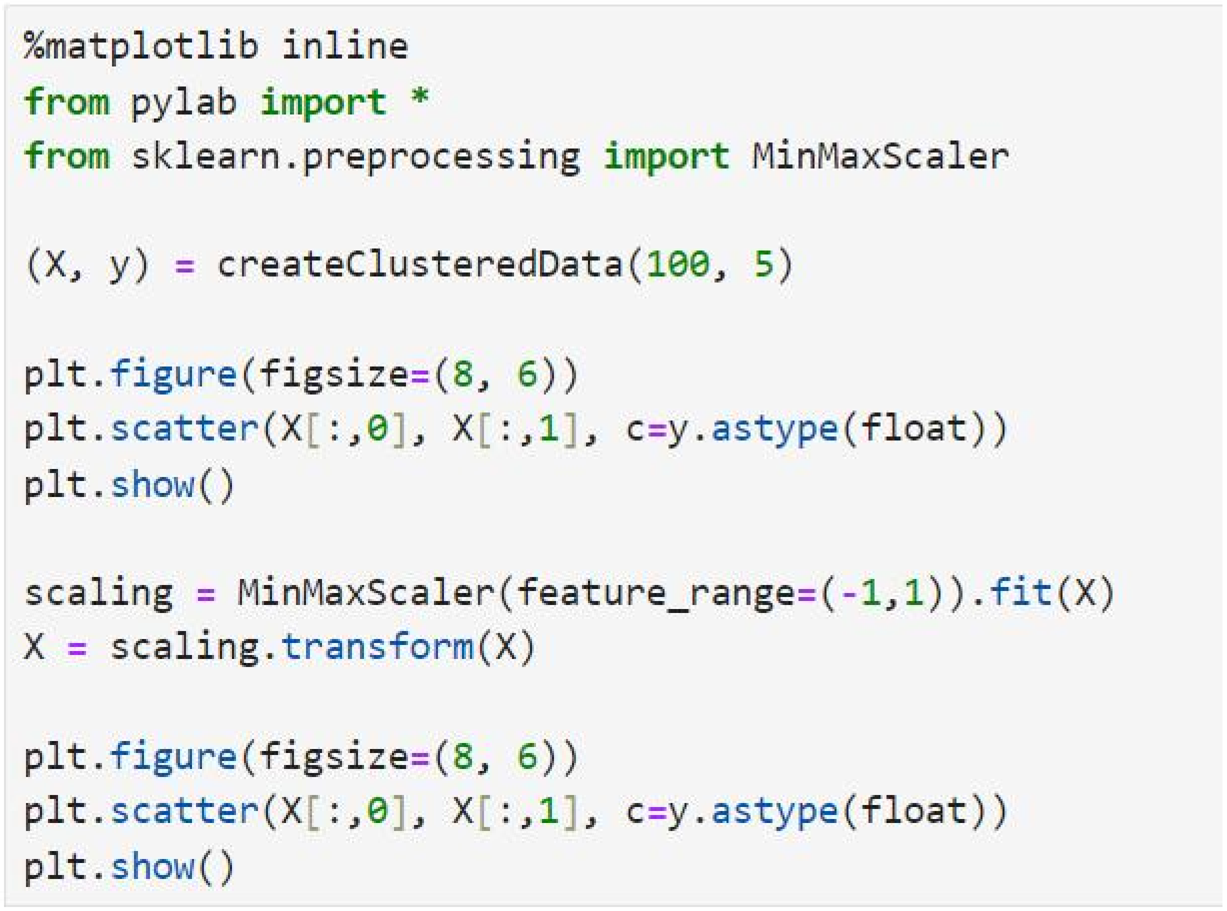
```
import sys
import matplotlib
matplotlib.use('Agg')

%matplotlib inline
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

df = pandas.read_csv("data2.csv")

d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)

features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]
y = df['Go']

dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)

tree.plot_tree(dtree, feature_names=features)

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



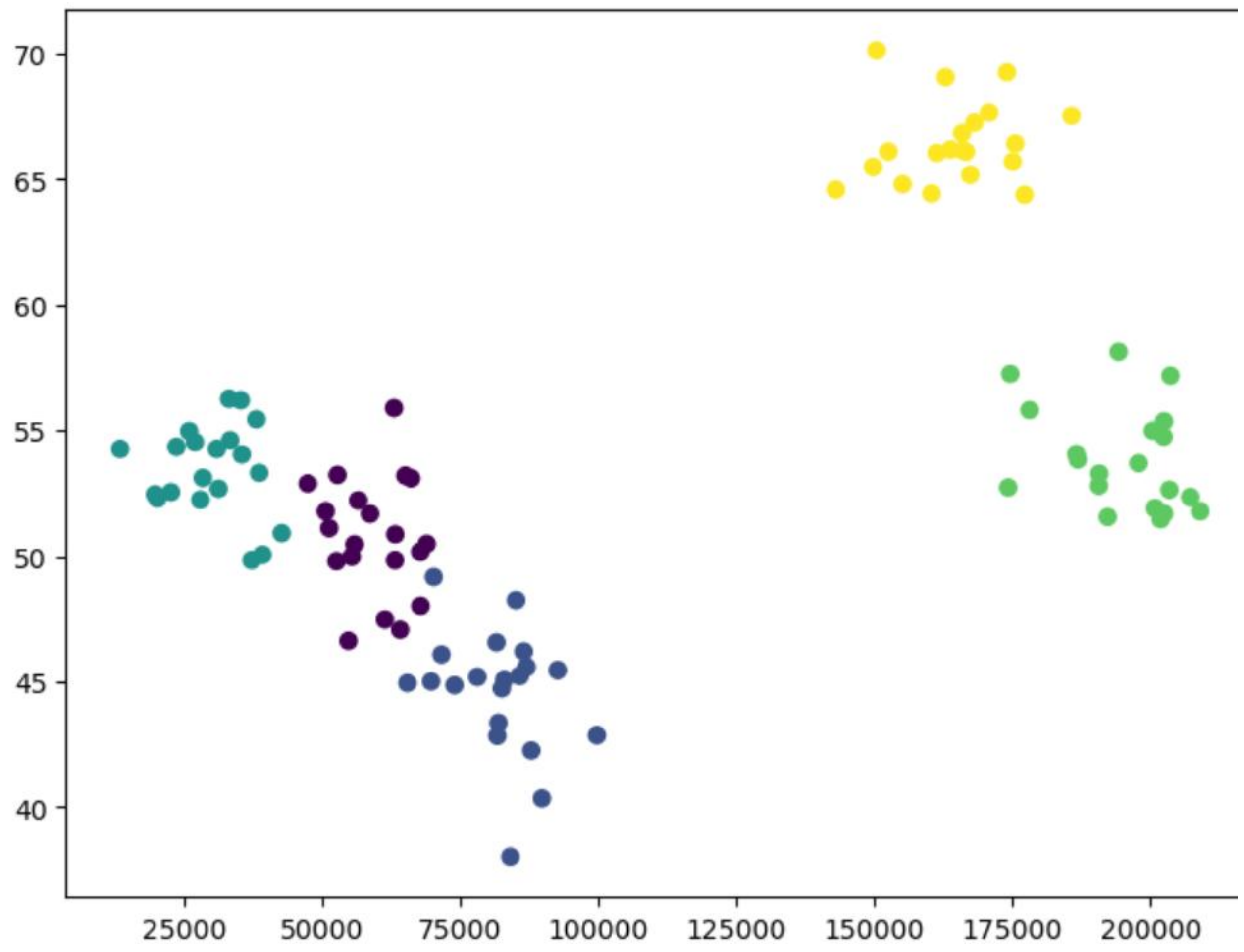
Week 13: Build model using SVM with different kernels

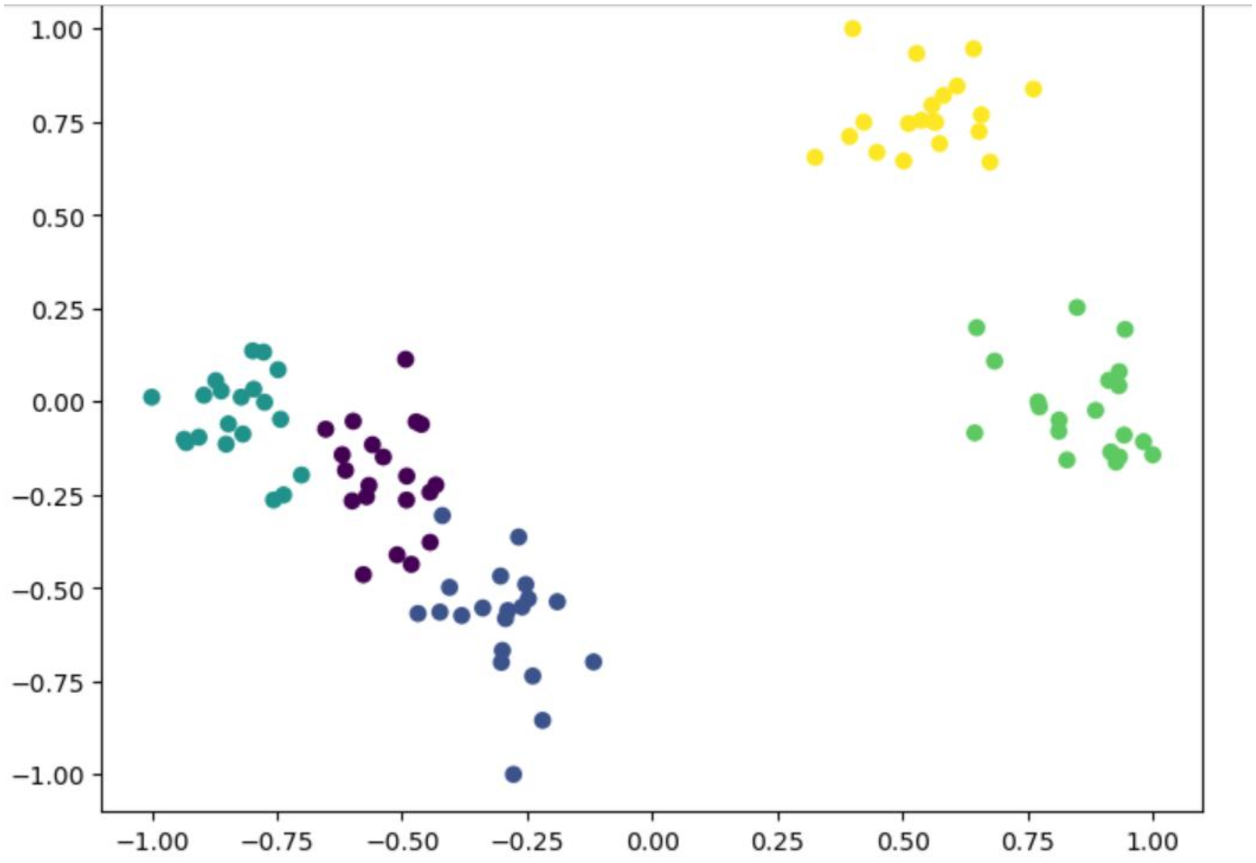


```
import numpy as np

#Create fake income/age clusters for N people in k clusters
def createClusteredData(N, k):
    np.random.seed(1234)
    pointsPerCluster = float(N)/k
    X = []
    y = []
    for i in range (k):
        incomeCentroid = np.random.uniform(20000.0, 200000.0)
        ageCentroid = np.random.uniform(20.0, 70.0)
        for j in range(int(pointsPerCluster)):
            X.append([np.random.normal(incomeCentroid, 10000.0), np.random.normal(ageCentroid, 2.0)])
            y.append(i)
    X = np.array(X)
    y = np.array(y)
    return X, y
```



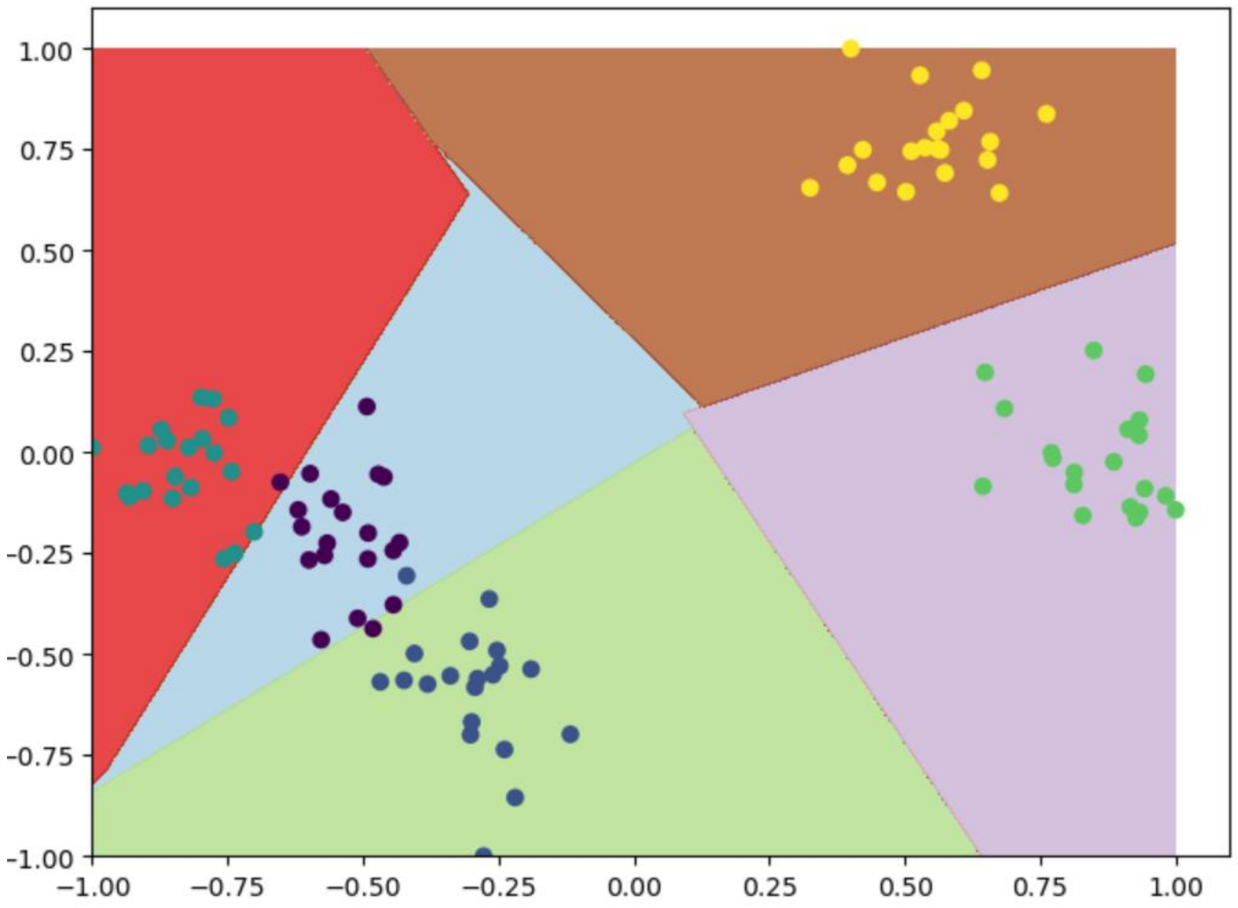
```
%matplotlib inline
from pylab import *
from sklearn.preprocessing import MinMaxScaler

(X, y) = createClusteredData(100, 5)

plt.figure(figsize=(8, 6))
plt.scatter(X[:,0], X[:,1], c=y.astype(float))
plt.show()

scaling = MinMaxScaler(feature_range=(-1,1)).fit(X)
X = scaling.transform(X)

plt.figure(figsize=(8, 6))
plt.scatter(X[:,0], X[:,1], c=y.astype(float))
plt.show()
```


70
65
60
55
50
45
40
25000
50000
75000
100000
125000
150000
175000
200000



```python
from sklearn import svm, datasets

C = 1.0
svc = svm.SVC(kernel='sigmoid', C=C).fit(X, y)
```

```python
def plotPredictions(clf):
    # Create a dense grid of points to sample
    xx, yy = np.meshgrid(np.arange(-1, 1, .001),
                     np.arange(-1, 1, .001))

    # Convert to Numpy arrays
    npx = xx.ravel()
    npy = yy.ravel()

    # Convert to a list of 2D (income, age) points
    samplePoints = np.c_[npx, npy]

    # Generate predicted labels (cluster numbers) for each point
    Z = clf.predict(samplePoints)

    plt.figure(figsize=(8, 6))
    Z = Z.reshape(xx.shape) #Reshape results to match xx dimension
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8) # Draw the contour
    plt.scatter(X[:,0], X[:,1], c=y.astype(float)) # Draw the points
    plt.show()

plotPredictions(svc)
```



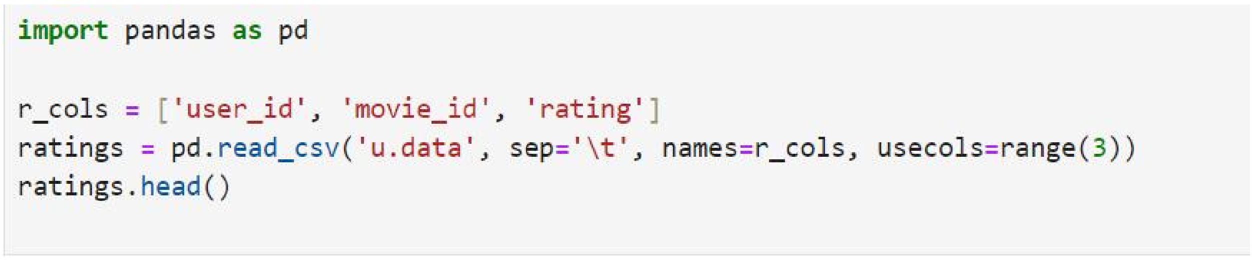
```
print(svc.predict(scaling.transform([[200000, 40]])))
```

```
[3]
```

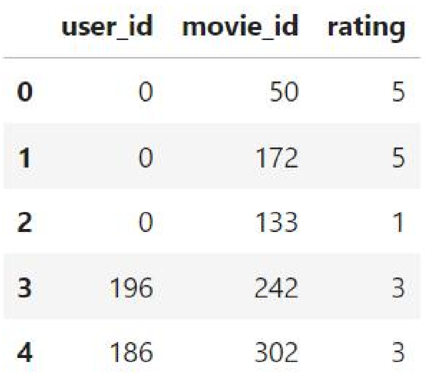
```
print(svc.predict(scaling.transform([[50000, 65]])))
```

```
[2]
```

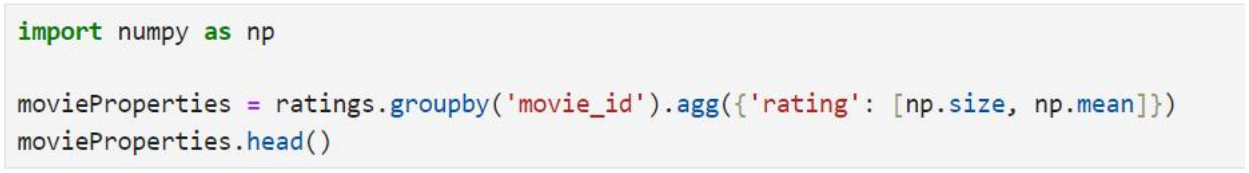
Week 14: Implement K-NN algorithm to classify a dataset.



```
import pandas as pd

r_cols = ['user_id', 'movie_id', 'rating']
ratings = pd.read_csv('u.data', sep='\t', names=r_cols, usecols=range(3))
ratings.head()
```



	user_id	movie_id	rating
0	0	50	5
1	0	172	5
2	0	133	1
3	196	242	3
4	186	302	3



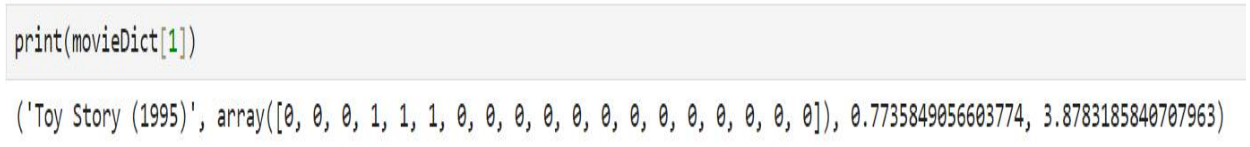
```
import numpy as np

movieProperties = ratings.groupby('movie_id').agg({'rating': [np.size, np.mean]})
movieProperties.head()
```

```
               rating
         size      mean
movie_id
       1  452  3.878319
       2  131  3.206107
       3   90  3.033333
       4  209  3.550239
       5   86  3.302326
```

```
movieNumRatings = pd.DataFrame(movieProperties['rating']['size'])
movieNormalizedNumRatings = movieNumRatings.apply(lambda x: (x - np.min(x)) / (np.max(x) - np.min(x)))
movieNormalizedNumRatings.head()
```

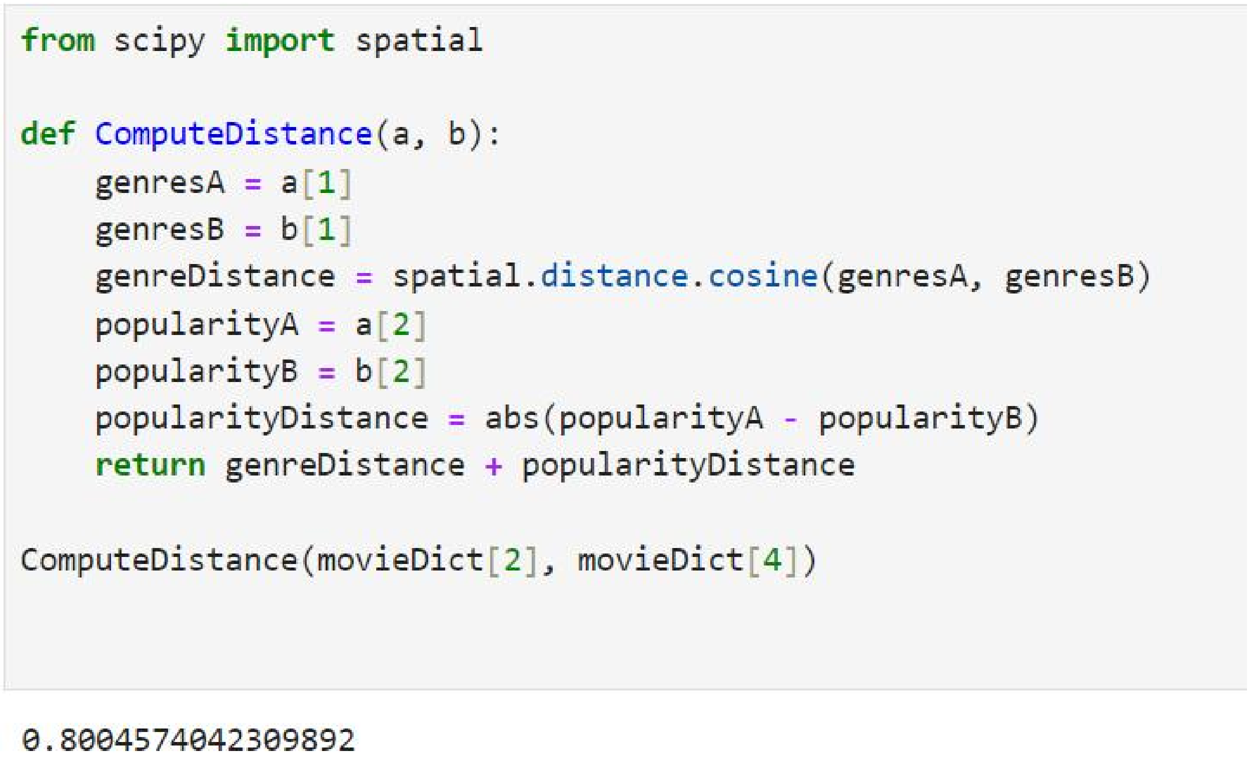
```
              size
movie_id
       1  0.773585
       2  0.222985
       3  0.152659
       4  0.356775
       5  0.145798
```

```
movieDict = {}
with open(r'u.item', encoding="ISO-8859-1") as f:
    temp = ''
    for line in f:
        #line.decode("ISO-8859-1")
        fields = line.rstrip('\n').split('|')
        movieID = int(fields[0])
        name = fields[1]
        genres = fields[5:25]
        genres = map(int, genres)
        movieDict[movieID] = (name, np.array(list(genres)), movieNormalizedNumRatings.loc[movieID].get('size'), movieProperties.loc[movieID].rating.get('
```



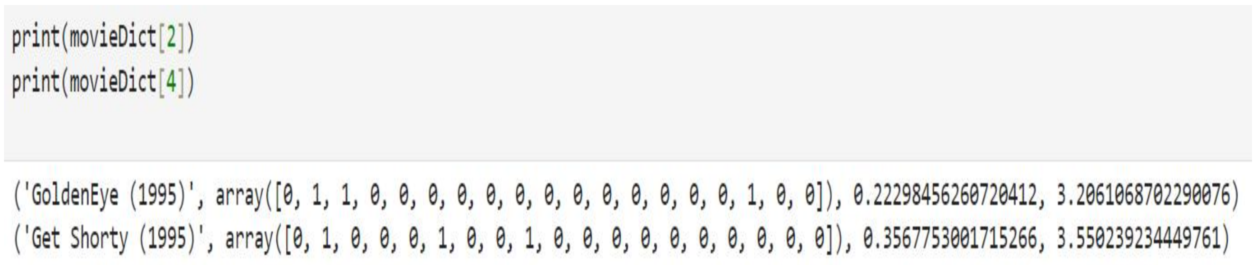
```python
print(movieDict[1])
```

```
('Toy Story (1995)', array([0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), 0.7735849056603774, 3.8783185840707963)
```



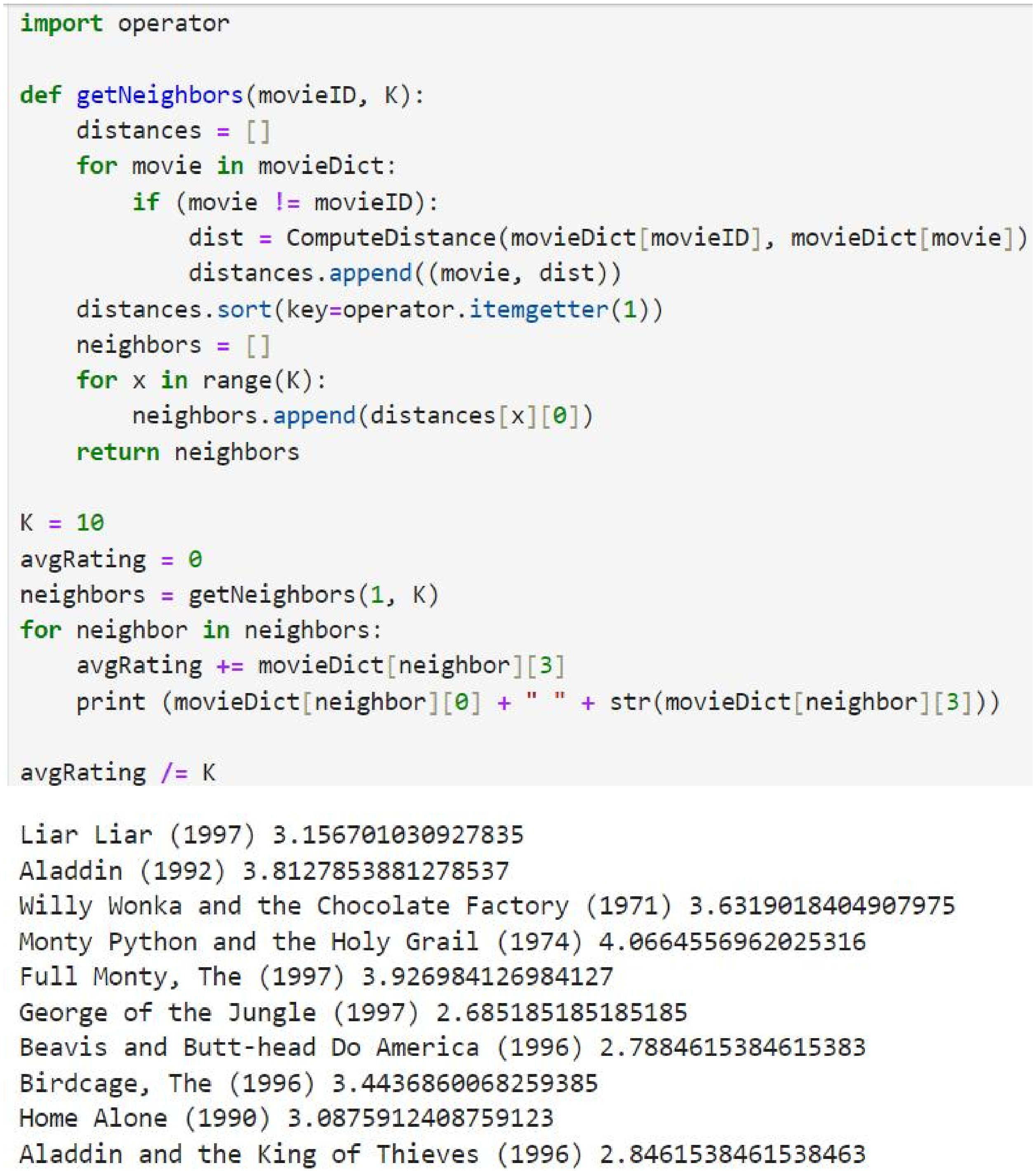
```python
from scipy import spatial

def ComputeDistance(a, b):
    genresA = a[1]
    genresB = b[1]
    genreDistance = spatial.distance.cosine(genresA, genresB)
    popularityA = a[2]
    popularityB = b[2]
    popularityDistance = abs(popularityA - popularityB)
    return genreDistance + popularityDistance

ComputeDistance(movieDict[2], movieDict[4])
```

```
0.8004574042309892
```



```python
print(movieDict[2])
print(movieDict[4])
```

```
('GoldenEye (1995)', array([0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]), 0.22298456260720412, 3.2061068702290076)
('Get Shorty (1995)', array([0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]), 0.3567753001715266, 3.550239234449761)
```



```python
import operator

def getNeighbors(movieID, K):
    distances = []
    for movie in movieDict:
        if (movie != movieID):
            dist = ComputeDistance(movieDict[movieID], movieDict[movie])
            distances.append((movie, dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(K):
        neighbors.append(distances[x][0])
    return neighbors

K = 10
avgRating = 0
neighbors = getNeighbors(1, K)
for neighbor in neighbors:
    avgRating += movieDict[neighbor][3]
    print (movieDict[neighbor][0] + " " + str(movieDict[neighbor][3]))

avgRating /= K
```

```
Liar Liar (1997) 3.156701030927835
Aladdin (1992) 3.8127853881278537
Willy Wonka and the Chocolate Factory (1971) 3.6319018404907975
Monty Python and the Holy Grail (1974) 4.0664556962025316
Full Monty, The (1997) 3.926984126984127
George of the Jungle (1997) 2.685185185185185
Beavis and Butt-head Do America (1996) 2.7884615384615383
Birdcage, The (1996) 3.4436860068259385
Home Alone (1990) 3.0875912408759123
Aladdin and the King of Thieves (1996) 2.8461538461538463
```

```
avgRating
```

```
3.3445905900235564
```

How does this compare to Toy Story's actual average rating?

```
movieDict[1]
```

```
('Toy Story (1995)',
 array([0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
 0.7735849056603774,
 3.8783185840707963)
```

Week 15: Build model to perform Clustering using K-means after applying PCA and determining the value of K using Elbow method.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#To view the graph in jupyter notebook
```

```
data = pd.read_csv('Mall_Customers.csv')
```

```
data.info()
```

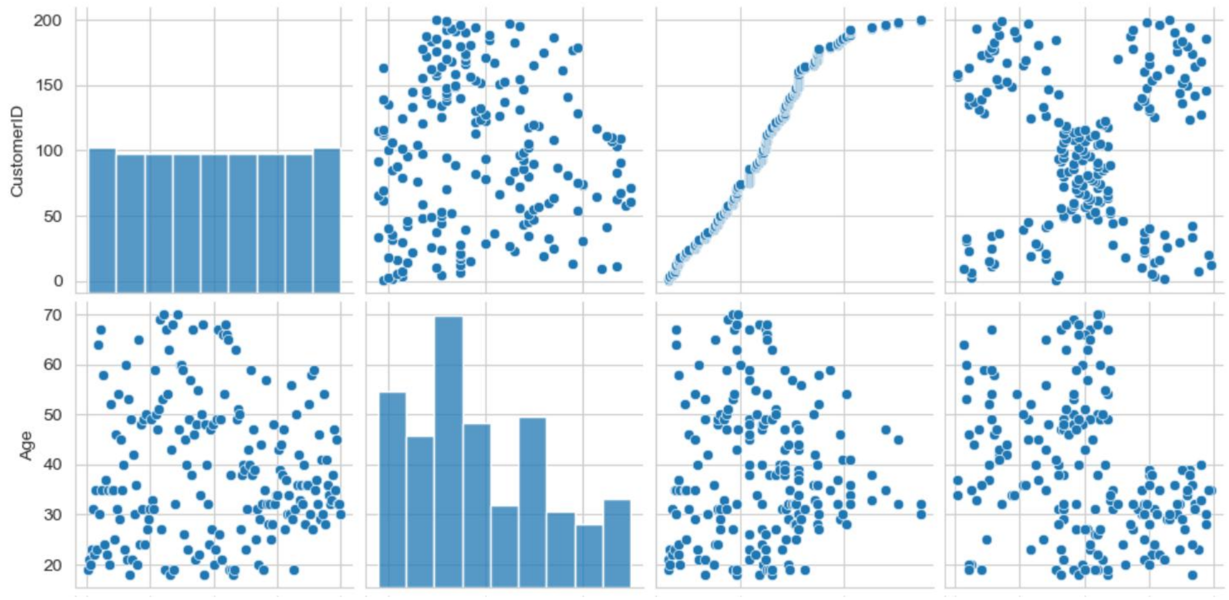
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64 
 3   Annual Income (k$)      200 non-null    int64 
 4   Spending Score (1-100)  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

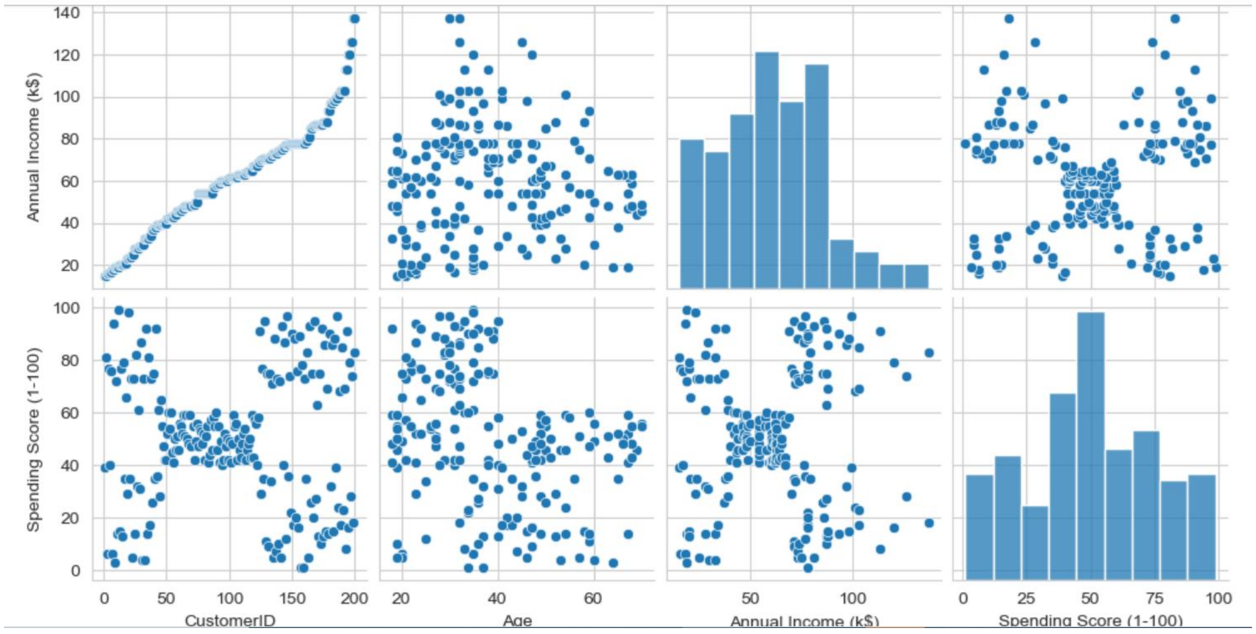
```
data.columns = ['CustomerID','Gender','Age','Annual Income (k$)','Spending Score (1-100)']
data.head()
```

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
sns.pairplot(data) #He will only plot numerical values. He will ignore non-numeric colns
```

```
<seaborn.axisgrid.PairGrid at 0x25d224c09e0>
```

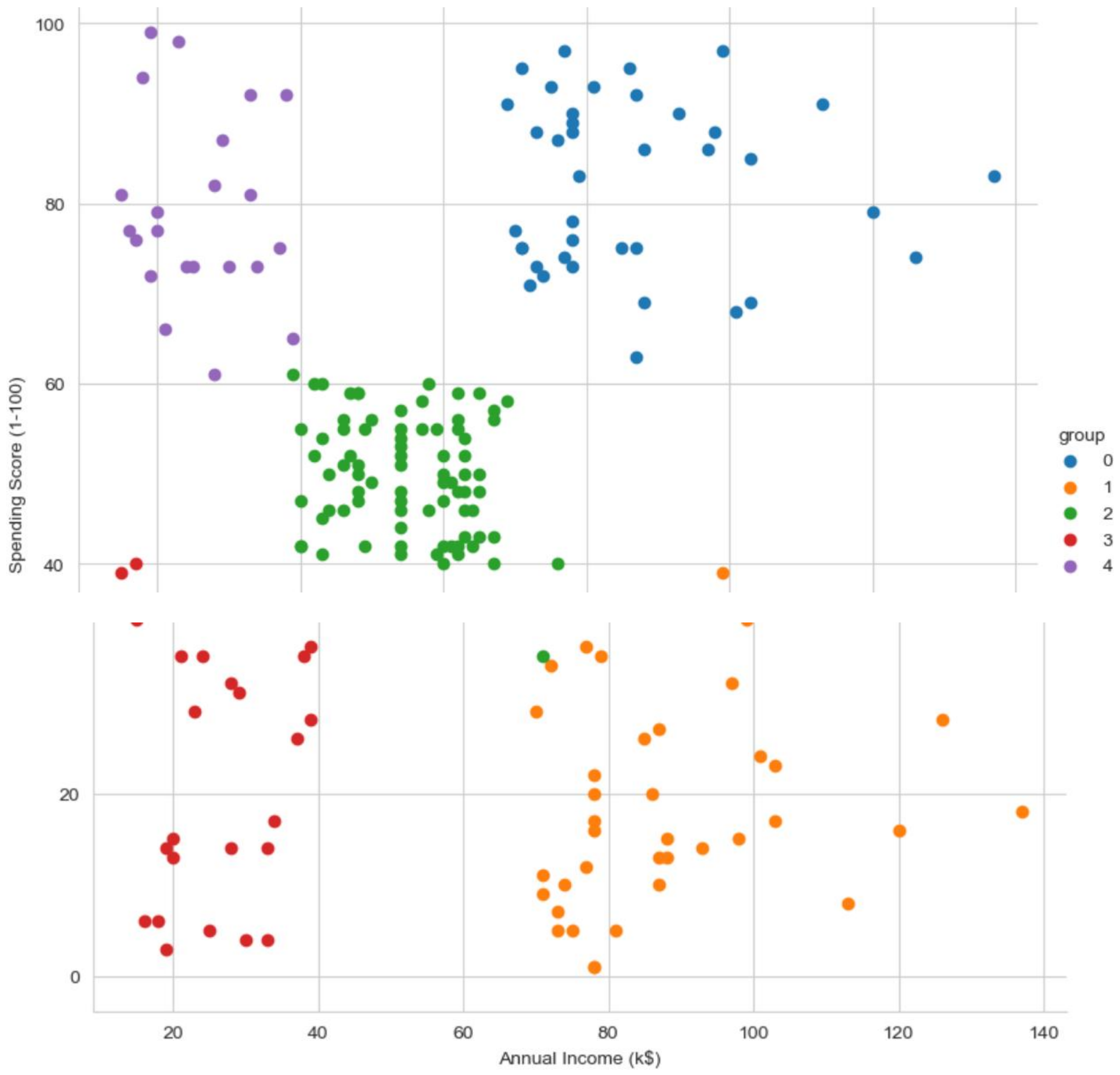




```
finalDataSet = data
finalDataSet['group']=group
finalDataSet.head(10)
```

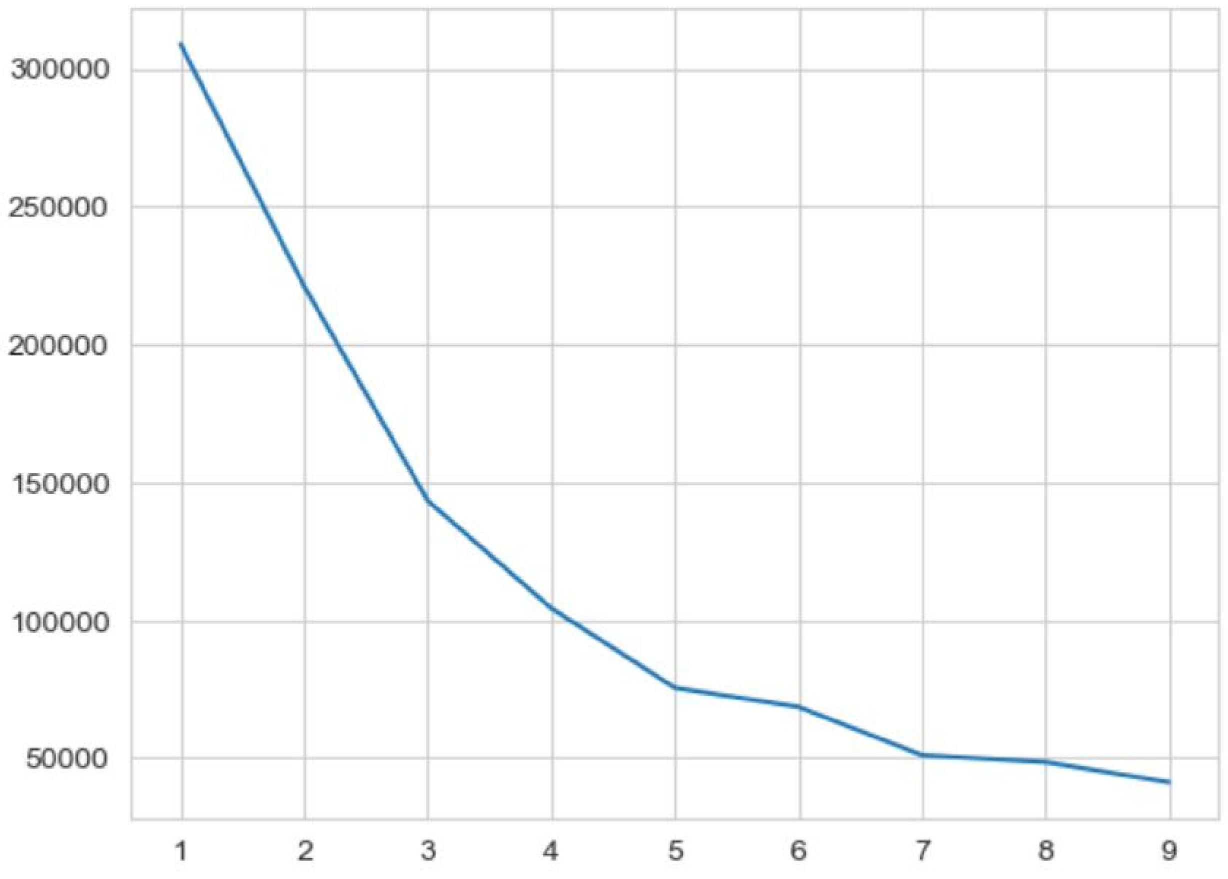
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	group
0	1	Male	19	15	39	3
1	2	Male	21	15	81	4
2	3	Female	20	16	6	3
3	4	Female	23	16	77	4
4	5	Female	31	17	40	3
5	6	Female	22	17	76	4
6	7	Female	35	18	6	3
7	8	Female	23	18	94	4
8	9	Male	64	19	3	3
9	10	Female	30	19	72	4

```
sns.set_style("whitegrid")
sns.FacetGrid(finalDataSet, hue="group", height= 8 ) \
.map(plt.scatter, "Annual Income (k$)", "Spending Score (1-100)") \
.add_legend();

#plt.show()
```



```
#You can verify whether the k was really ideal or not !!!

#Elbow Method
# For each k value we will plot the error rate(misclassification rate)
# using a formula (WCSS - WithIn Cluster Sum of Squares)
features= data.iloc[:,[2,3,4]]
OMP_NUM_THREADS = 1
from sklearn.cluster import KMeans
errorValues = []

for i in range(1,10):
    m1 = KMeans(n_clusters=i)
    m1.fit(features)
    errorValues.append(m1.inertia_)

plt.plot(range(1,10) , errorValues)
```

```
[<matplotlib.lines.Line2D at 0x25d23406f90>]
```


300000
250000
200000
150000
100000
50000
1
2
3
4
5
6
7
8
9