Python and Deep Learning Programming

CS 5590

LAB-1 Report

Bala Vignan Reddy Kandula

Class ID: 9

bkr62@mail.umkc.edu

Vidyullatha Lakshmi Kaza

Class ID: 13

vrkwc@mail.umkc.edu

Vikas Narravula

Class ID: 27

vnzf8@mail.umkc.edu

GitHub Link: https://github.com/VidyullathaKaza/CS5590-Python_DeepLearning_Labs/wiki/Lab-1

YouTube Link:

Part-1: **https://youtu.be/_-fJOc4tkVM**

Part-2: https://youtu.be/JQSmQFPxxZM

Question-1

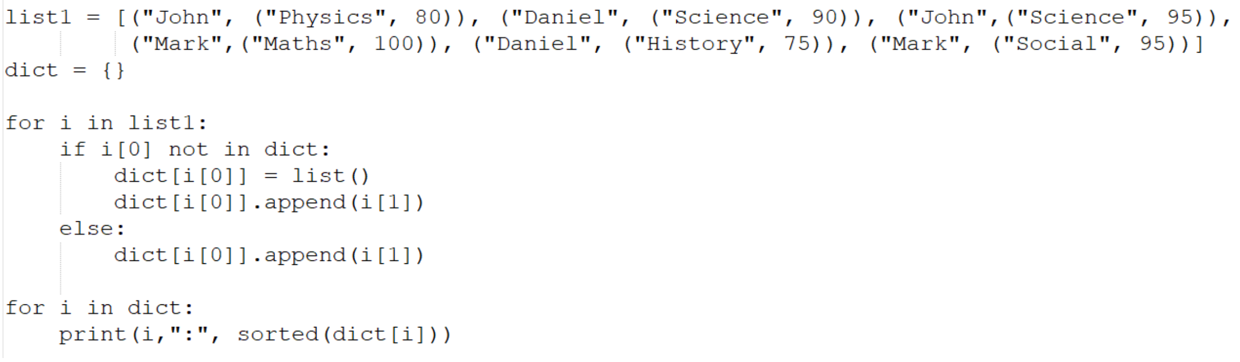
1. Suppose you have a list of tuples as follows:

[(‘John’, (‘Physics’, 80)) , (‘ Daniel’, (‘Science’, 90)), (‘John’, (‘Science’, 95)), (‘Mark’,(‘Maths’, 100)), (‘Daniel’, (’History’, 75)), (‘Mark’, (‘Social’, 95))]

Create a dictionary with keys as names and values as list of (subjects, marks) in sorted order.

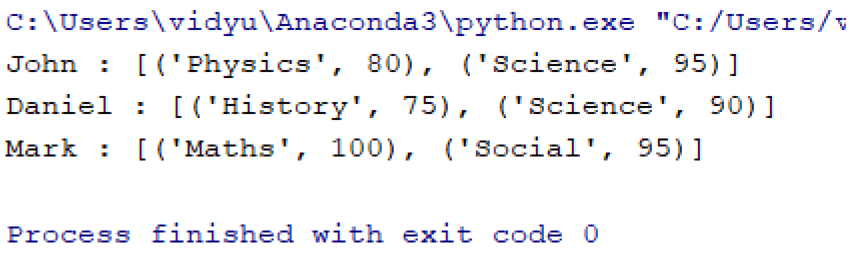
```
{
    John : [(‘Physics’, 80), (‘Science’, 95)]
    Daniel : [ (’History’, 75), (‘Science’, 90)]
    Mark : [ (‘Maths’, 100), (‘Social’, 95)]
}
```

We created a list named list1 with our input values and created an empty dictionary. If the name of the person was not a key in the dictionary, I created a new person and empty list. I then appended to that tuple. If the name is already in the list then I simply added the class and grade to that key.



```
list1 = [("John", ("Physics", 80)), ("Daniel", ("Science", 90)), ("John",("Science", 95)),
         ("Mark",("Maths", 100)), ("Daniel", ("History", 75)), ("Mark", ("Social", 95))]
dict = {}

for i in list1:
    if i[0] not in dict:
        dict[i[0]] = list()
        dict[i[0]].append(i[1])
    else:
        dict[i[0]].append(i[1])

for i in dict:
    print(i,":", sorted(dict[i]))
```

Final step was to print the dictionary in the sorted order:

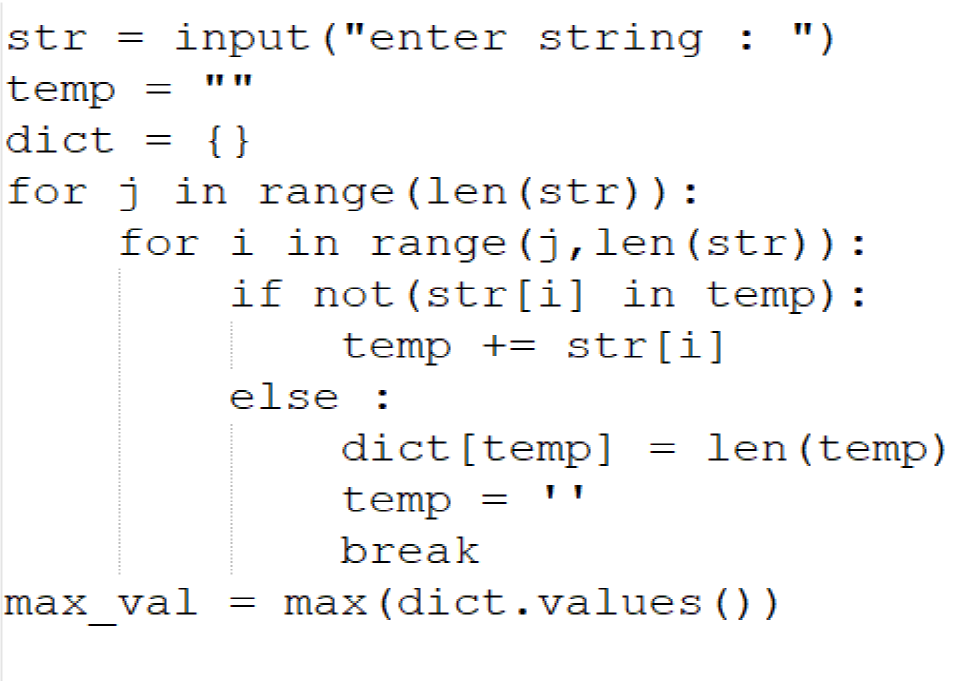


```
C:\Users\vidyu\Anaconda3\python.exe "C:/Users/v
John : [('Physics', 80), ('Science', 95)]
Daniel : [('History', 75), ('Science', 90)]
Mark : [('Maths', 100), ('Social', 95)]

Process finished with exit code 0
```

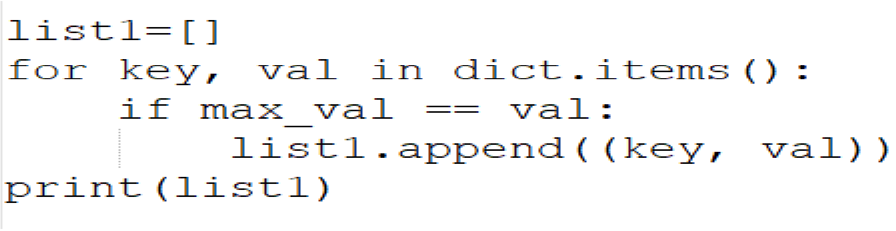
Question-2

Given a string, find the longest substrings without repeating characters along with the length as a tuple

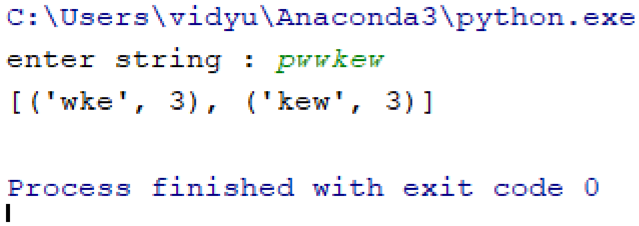
Input: "pwwkew"
Output: (wke,3), (kew,3)



```
str = input("enter string : ")
temp = ""
dict = {}
for j in range(len(str)):
    for i in range(j,len(str)):
        if not(str[i] in temp):
            temp += str[i]
        else :
            dict[temp] = len(temp)
            temp = ''
            break
max_val = max(dict.values())
```



```
list1=[]
for key, val in dict.items():
    if max_val == val:
        list1.append((key, val))
print(list1)
```



```
C:\Users\vidyu\Anaconda3\python.exe
enter string : pwwkew
[('wke', 3), ('kew', 3)]

Process finished with exit code 0
```

The Basic idea of solution is like to determine if a string has all unique characters, and count the to the list.

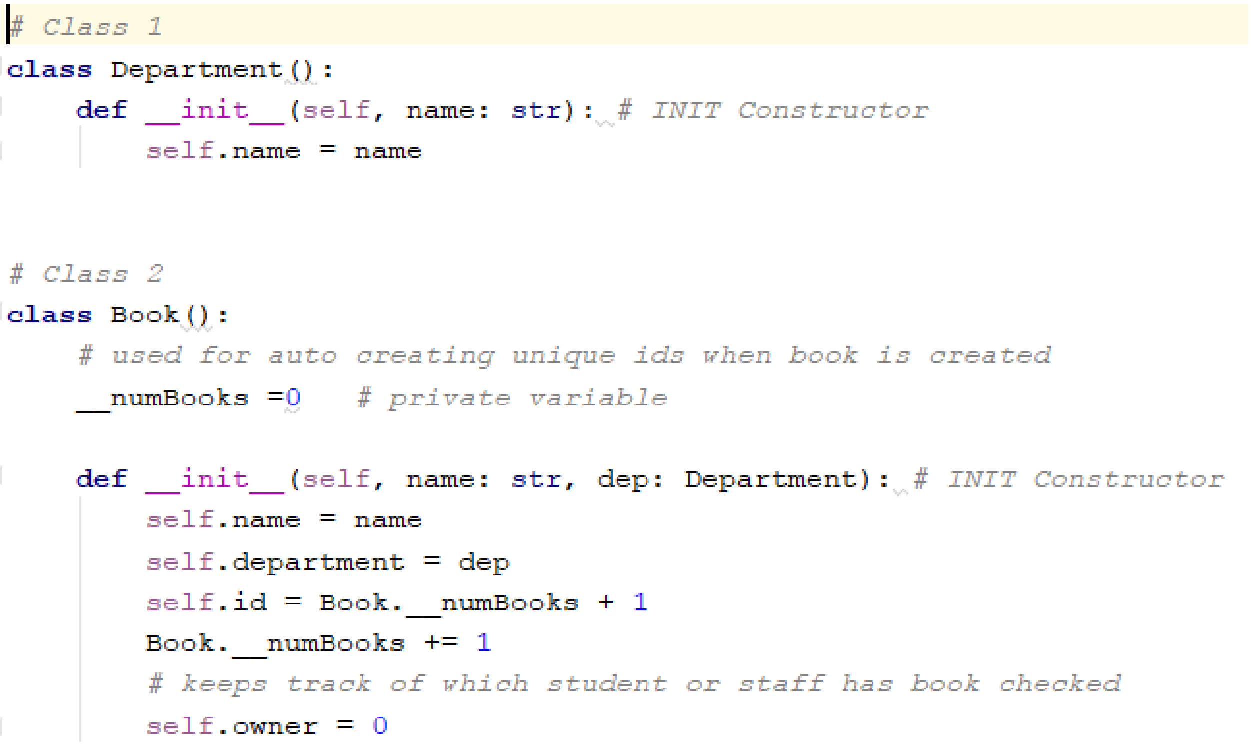
Question-3

3. Write a python program to create any one of the following management systems.
   1. Airline Booking Reservation System (e.g. classes Flight, Person, Employee, Passenger etc.)
   2. Library Management System (eg: Student, Book, Faculty, Department etc.)

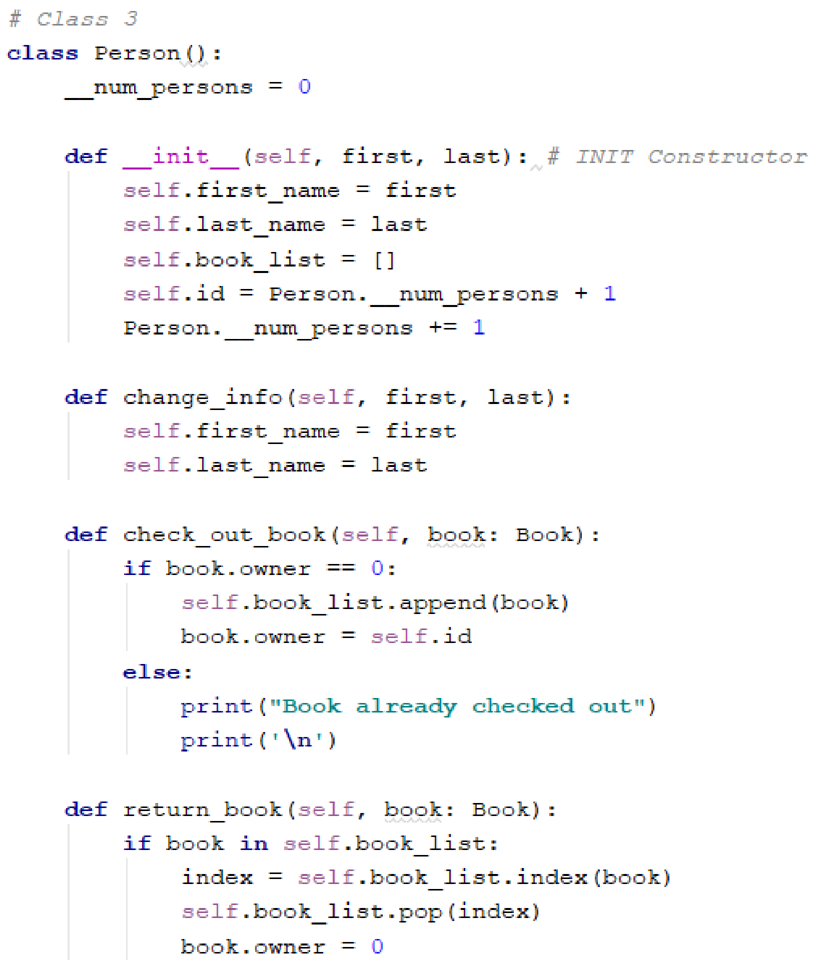
Prerequisites:
a. Your code should show inheritance at least once
b. Your code should have one super call
c. Use at least one private data member in your code
d. Create instances of all classes and show the relationship between them
Comment your code appropriately to point out where all these things are present

Library Management System

Now, we have created the basic required classes class 1 and class 2 named department and book.

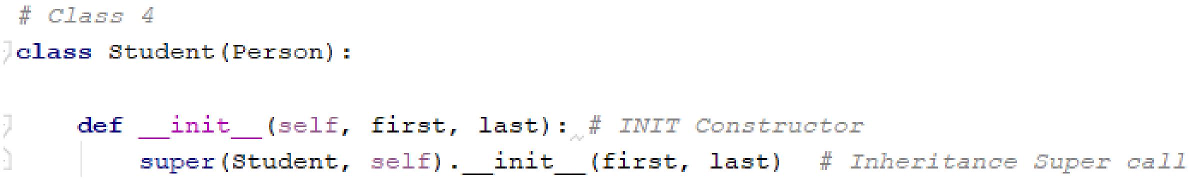


```
# Class 1
class Department():
    def __init__(self, name: str): # INIT Constructor
        self.name = name


# Class 2
class Book():
    # used for auto creating unique ids when book is created
    __numBooks =0    # private variable

    def __init__(self, name: str, dep: Department): # INIT Constructor
        self.name = name
        self.department = dep
        self.id = Book.__numBooks + 1
        Book.__numBooks += 1
        # keeps track of which student or staff has book checked
        self.owner = 0
```

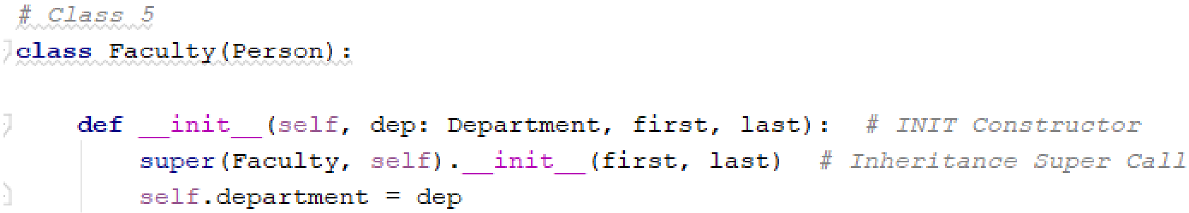


```
# Class 3
class Person():
    __num_persons = 0

    def __init__(self, first, last): # INIT Constructor
        self.first_name = first
        self.last_name = last
        self.book_list = []
        self.id = Person.__num_persons + 1
        Person.__num_persons += 1

    def change_info(self, first, last):
        self.first_name = first
        self.last_name = last

    def check_out_book(self, book: Book):
        if book.owner == 0:
            self.book_list.append(book)
            book.owner = self.id
        else:
            print("Book already checked out")
            print('\n')

    def return_book(self, book: Book):
        if book in self.book_list:
            index = self.book_list.index(book)
            self.book_list.pop(index)
            book.owner = 0
```

In person class, we created functions for check out books and return books. We created a private variable named __num_persons. The book list of person class of our code includes checked out as an element, it tracks the id of the person who took it. By this way we can know the current holder of the book and what books are checked out under a person.

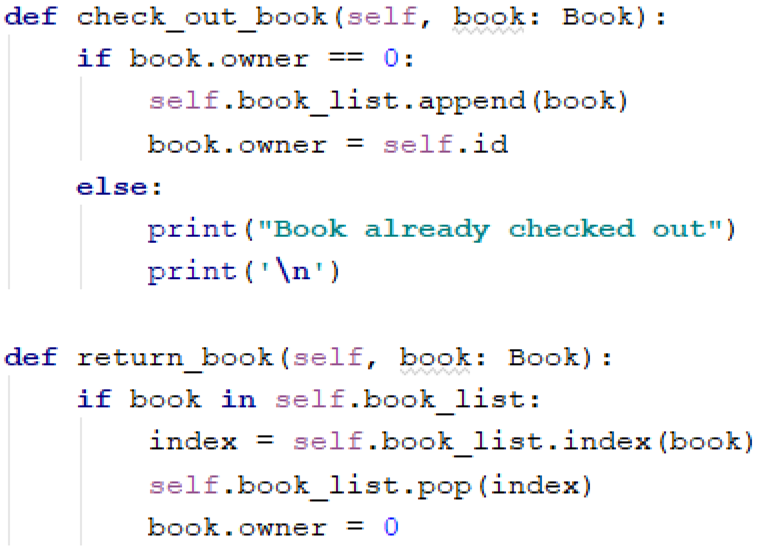


```
# Class 4
class Student(Person):

    def __init__(self, first, last): # INIT Constructor
        super(Student, self).__init__(first, last)  # Inheritance Super call
```



```
# Class 5
class Faculty(Person):

    def __init__(self, dep: Department, first, last):  # INIT Constructor
        super(Faculty, self).__init__(first, last)  # Inheritance Super Call
        self.department = dep
```

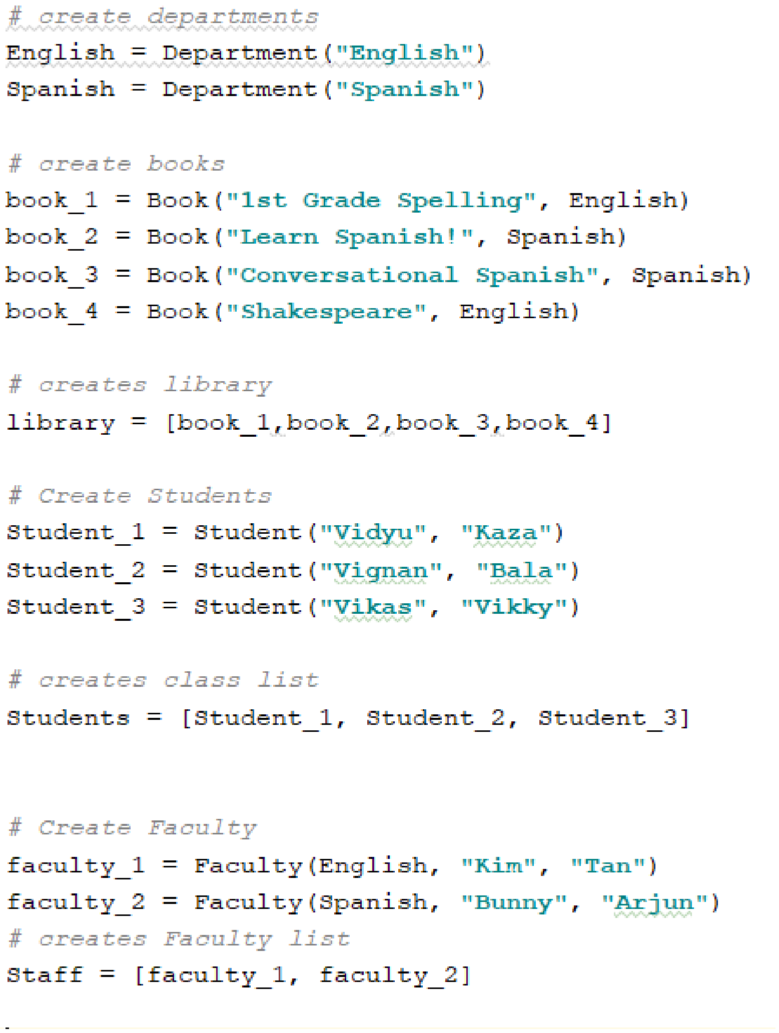
Here we used owner variable to prevent double checkout. If this variable is zero it indicates that no person is holding the book and if it is not 0 we can understand that book was checked out.



```
def check_out_book(self, book: Book):
    if book.owner == 0:
        self.book_list.append(book)
        book.owner = self.id
    else:
        print("Book already checked out")
        print('\n')

def return_book(self, book: Book):
    if book in self.book_list:
        index = self.book_list.index(book)
        self.book_list.pop(index)
        book.owner = 0
```

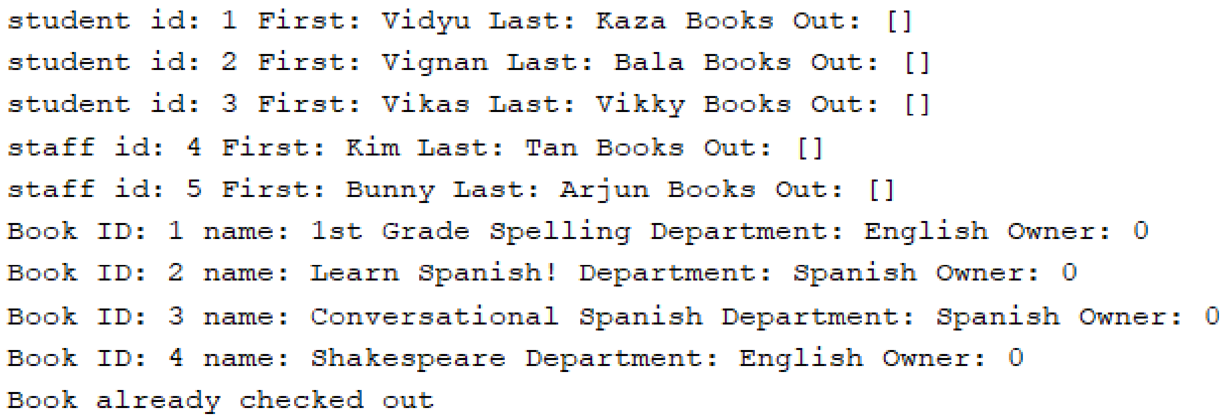
Test Output:

In order to test, I created multiple instances of the classes that were used for testing.



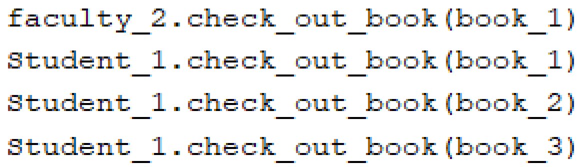
```
# create departments
English = Department("English")
Spanish = Department("Spanish")

# create books
book_1 = Book("1st Grade Spelling", English)
book_2 = Book("Learn Spanish!", Spanish)
book_3 = Book("Conversational Spanish", Spanish)
book_4 = Book("Shakespeare", English)

# creates library
library = [book_1,book_2,book_3,book_4]

# Create Students
Student_1 = Student("Vidyu", "Kaza")
Student_2 = Student("Vignan", "Bala")
Student_3 = Student("Vikas", "Vikky")

# creates class list
Students = [Student_1, Student_2, Student_3]


# Create Faculty
faculty_1 = Faculty(English, "Kim", "Tan")
faculty_2 = Faculty(Spanish, "Bunny", "Arjun")
# creates Faculty list
Staff = [faculty_1, faculty_2]
```

By observing below results, we can say that the objects were created correctly.



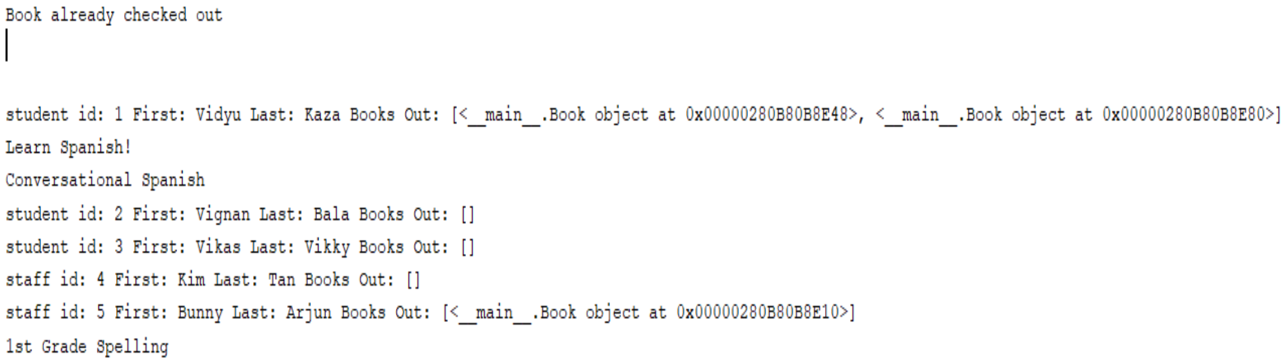
```
student id: 1 First: Vidyu Last: Kaza Books Out: []
student id: 2 First: Vignan Last: Bala Books Out: []
student id: 3 First: Vikas Last: Vikky Books Out: []
staff id: 4 First: Kim Last: Tan Books Out: []
staff id: 5 First: Bunny Last: Arjun Books Out: []
Book ID: 1 name: 1st Grade Spelling Department: English Owner: 0
Book ID: 2 name: Learn Spanish! Department: Spanish Owner: 0
Book ID: 3 name: Conversational Spanish Department: Spanish Owner: 0
Book ID: 4 name: Shakespeare Department: English Owner: 0
Book already checked out
```

Then, I used the classes to check out some books:



```
faculty_2.check_out_book(book_1)
Student_1.check_out_book(book_1)
Student_1.check_out_book(book_2)
Student_1.check_out_book(book_3)
```

Notice that faculty_2 and Student_1 are both trying to check out book_1. This should prevent student 1 from obtaining the book because the faculty already has it. Notice that we get a log of an error "book already checked out"



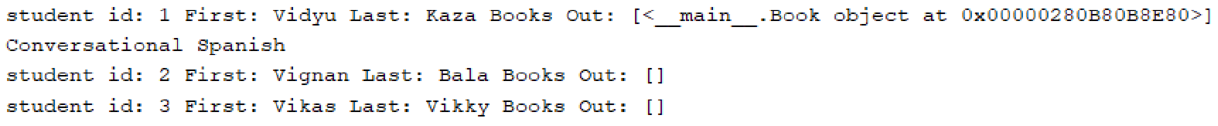
```
Book already checked out
|

student id: 1 First: Vidyu Last: Kaza Books Out: [<__main__.Book object at 0x00000280B80B8E48>, <__main__.Book object at 0x00000280B80B8E80>]
Learn Spanish!
Conversational Spanish
student id: 2 First: Vignan Last: Bala Books Out: []
student id: 3 First: Vikas Last: Vikky Books Out: []
staff id: 4 First: Kim Last: Tan Books Out: []
staff id: 5 First: Bunny Last: Arjun Books Out: [<__main__.Book object at 0x00000280B80B8E10>]
1st Grade Spelling
```

Student 1 was successful in getting book 2 and 3 and that Faculty 1 was able to get book_1.

We tested the return functionality:

```
Student_1.return_book(book_2)
```

After running this code could see that the book was successfully returned



```
student id: 1 First: Vidyu Last: Kaza Books Out: [<__main__.Book object at 0x00000280B80B8E80>]
Conversational Spanish
student id: 2 First: Vignan Last: Bala Books Out: []
student id: 3 First: Vikas Last: Vikky Books Out: []
```

Student 1 no longer had book_2 but still had book_3

Airline Reservation System

We created 5 classes named Flight, Person, Flights, Passenger, Pilot.

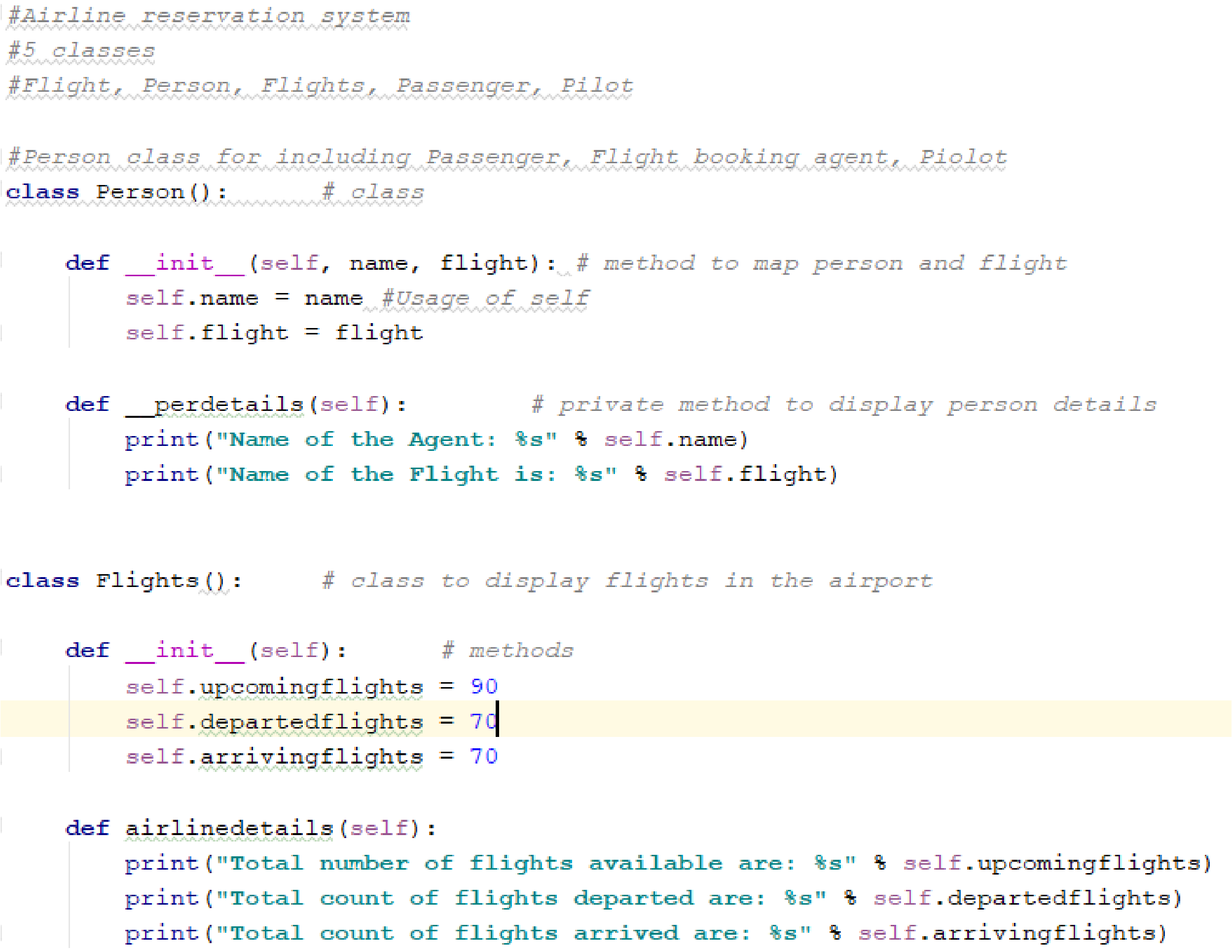
Class1: Person class includes Passenger, Flight booking agent, Pilot.

Class2: Flights class to display flights in the airport

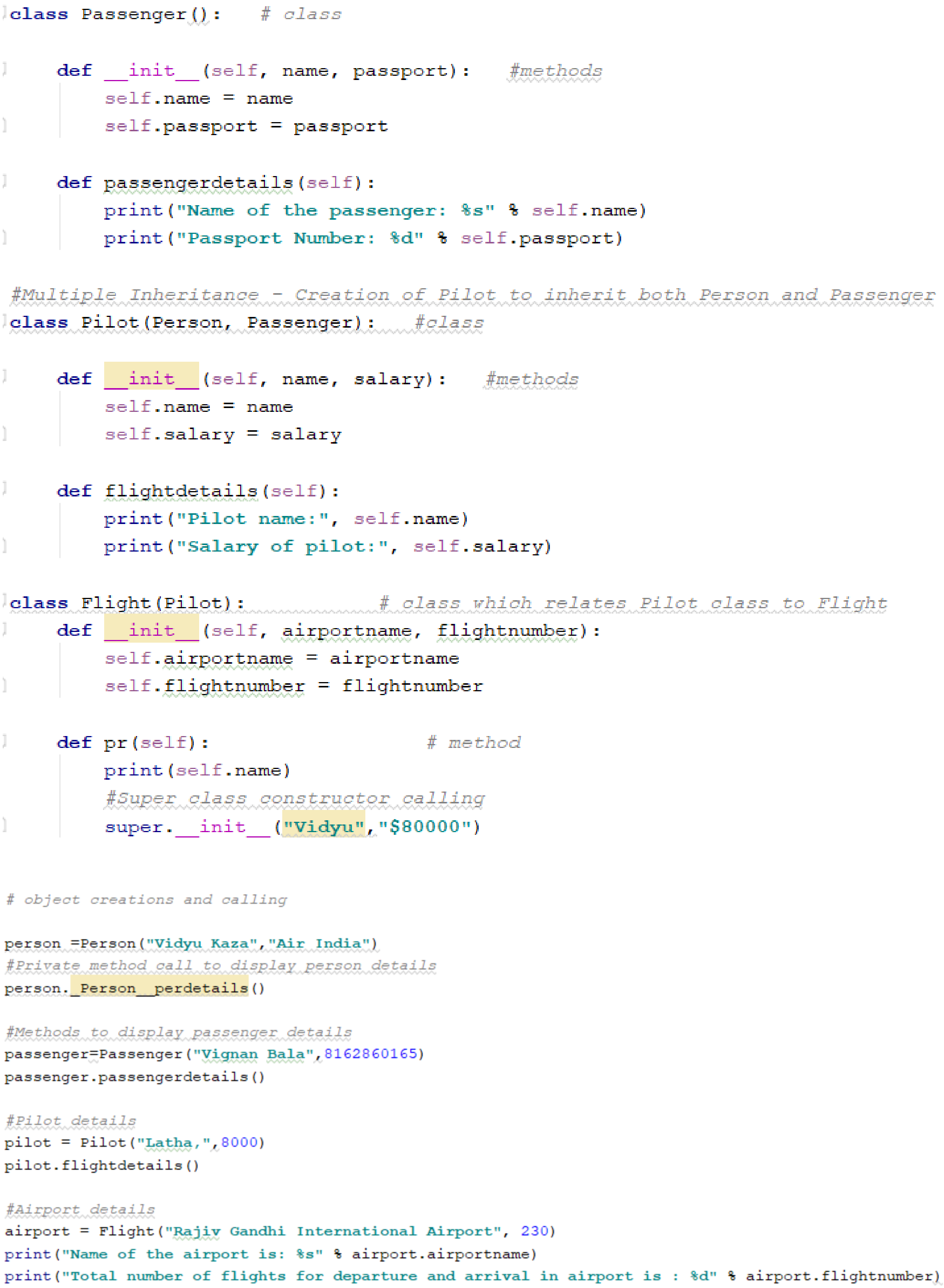
Class3: Passenger class includes passenger name and Passport number

Class4: Pilot class does multiple inherit both person and passenger classes

Class5: Flight class inherits pilot



```
#Airline reservation system
#5 classes
#Flight, Person, Flights, Passenger, Pilot

#Person class for including Passenger, Flight booking agent, Piolot
class Person():        # class

    def __init__(self, name, flight): # method to map person and flight
        self.name = name #Usage of self
        self.flight = flight

    def __perdetails(self):         # private method to display person details
        print("Name of the Agent: %s" % self.name)
        print("Name of the Flight is: %s" % self.flight)


class Flights():     # class to display flights in the airport

    def __init__(self):       # methods
        self.upcomingflights = 90
        self.departedflights = 70
        self.arrivingflights = 70

    def airlinedetails(self):
        print("Total number of flights available are: %s" % self.upcomingflights)
        print("Total count of flights departed are: %s" % self.departedflights)
        print("Total count of flights arrived are: %s" % self.arrivingflights)
```



```
class Passenger():    # class

    def __init__(self, name, passport):    #methods
        self.name = name
        self.passport = passport

    def passengerdetails(self):
        print("Name of the passenger: %s" % self.name)
        print("Passport Number: %d" % self.passport)

#Multiple Inheritance - Creation of Pilot to inherit both Person and Passenger
class Pilot(Person, Passenger):    #class

    def __init__(self, name, salary):    #methods
        self.name = name
        self.salary = salary

    def flightdetails(self):
        print("Pilot name:", self.name)
        print("Salary of pilot:", self.salary)

class Flight(Pilot):            # class which relates Pilot class to Flight
    def __init__(self, airportname, flightnumber):
        self.airportname = airportname
        self.flightnumber = flightnumber

    def pr(self):                  # method
        print(self.name)
        #Super class constructor calling
        super.__init__("Vidyu","$80000")


# object creations and calling

person =Person("Vidyu Kaza","Air India")
#Private method call to display person details
person._Person__perdetails()

#Methods to display passenger details
passenger=Passenger("Vignan Bala",8162860165)
passenger.passengerdetails()

#Pilot details
pilot = Pilot("Latha,",8000)
pilot.flightdetails()

#Airport details
airport = Flight("Rajiv Gandhi International Airport", 230)
print("Name of the airport is: %s" % airport.airportname)
print("Total number of flights for departure and arrival in airport is : %d" % airport.flightnumber)
```

We designed using the classes and constructors. The code consists of five classes.

The *init* constructor in all the classes are defined.

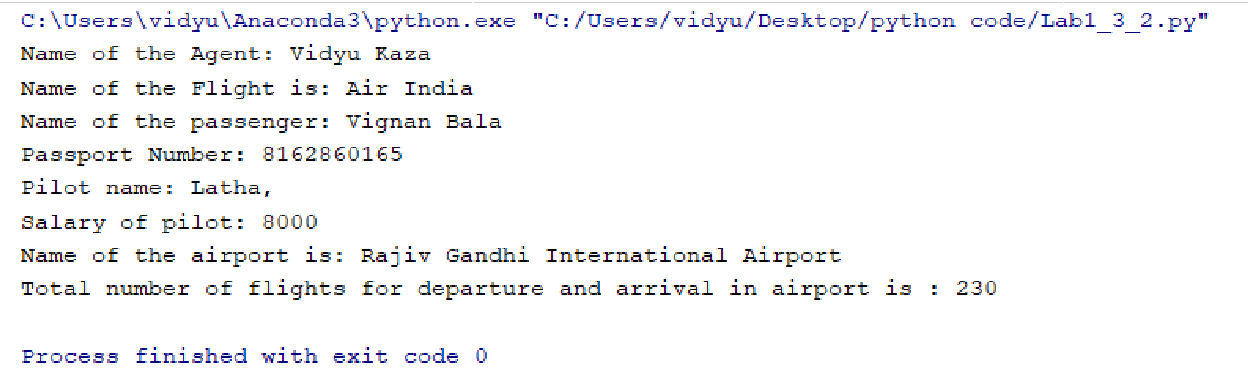
Inheritance was implemented.

Super call of method is done.

Self has been written throughout the question.

One private data member in the code has been written.

Multiple inheritance and relationship between the classes has been shown.



```
C:\Users\vidyu\Anaconda3\python.exe "C:/Users/vidyu/Desktop/python code/Lab1_3_2.py"
Name of the Agent: Vidyu Kaza
Name of the Flight is: Air India
Name of the passenger: Vignan Bala
Passport Number: 8162860165
Pilot name: Latha,
Salary of pilot: 8000
Name of the airport is: Rajiv Gandhi International Airport
Total number of flights for departure and arrival in airport is : 230

Process finished with exit code 0
```

The details of the agent, Flight and the other details as per the classes has been displayed. The airport name and the details of the number of departure and arrival flights are also displayed as per the user.

Question-4

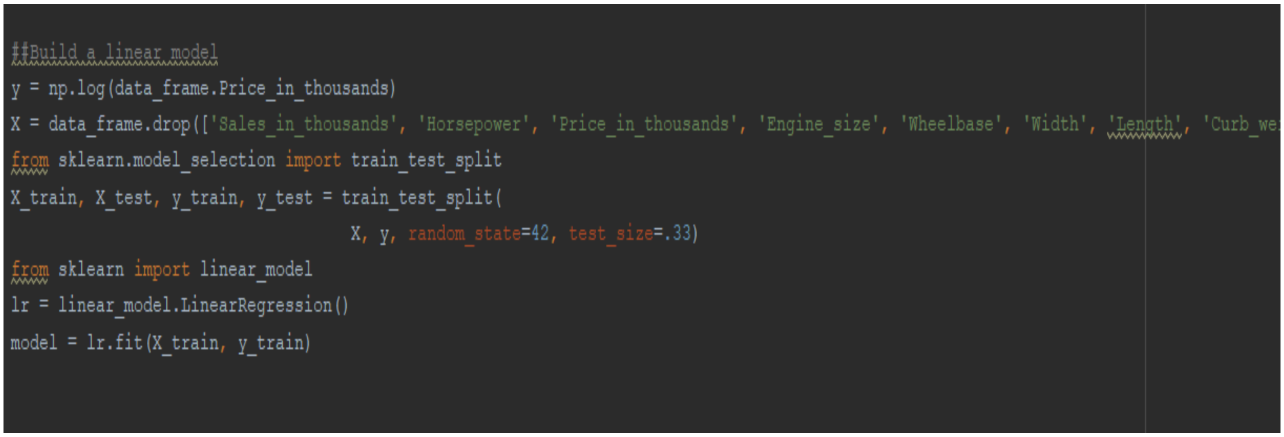
4. Create Multiple Regression by choosing a dataset of your choice (again before evaluating, clean the data set with the EDA learned in the class). Evaluate the model using RMSE and R2 and also report if you saw any improvement before and after the EDA.

The Dataset I choose is scikit-learn.org

Link:

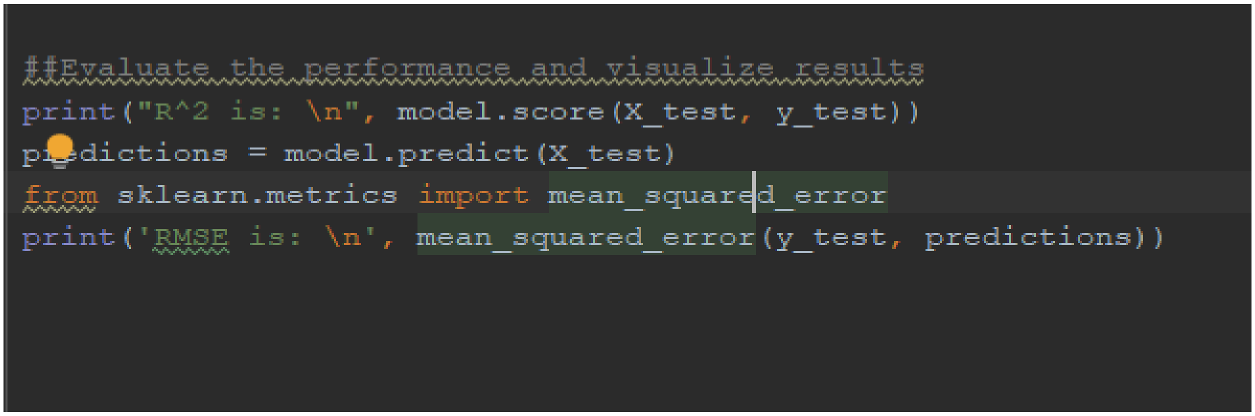
https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes

Split the dataset to training set and test set



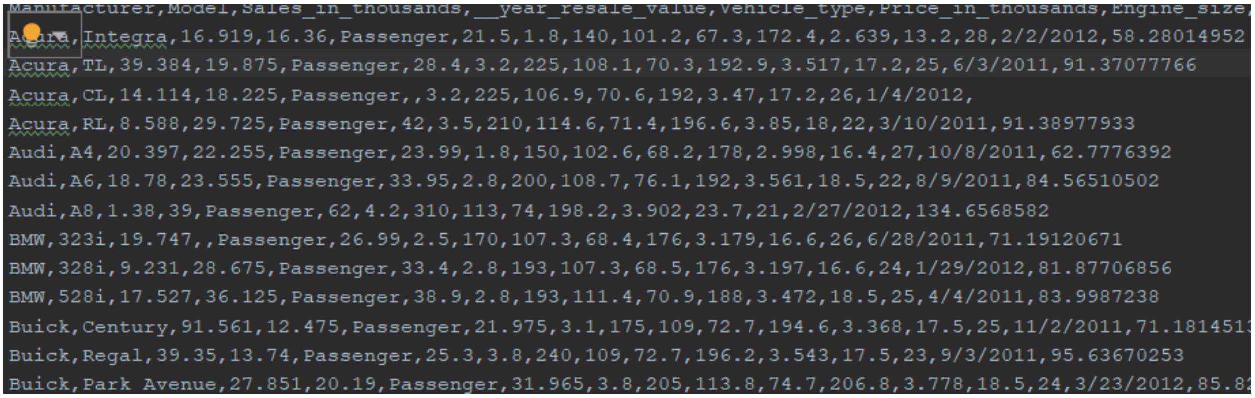
```
##Build a linear model
y = np.log(data_frame.Price_in_thousands)
X = data_frame.drop(['Sales_in_thousands', 'Horsepower', 'Price_in_thousands', 'Engine_size', 'Wheelbase', 'Width', 'Length', 'Curb_we
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
                                X, y, random_state=42, test_size=.33)
from sklearn import linear_model
lr = linear_model.LinearRegression()
model = lr.fit(X_train, y_train)
```

Evaluate the model using R^2 and RMSE Without EDA



```
##Evaluate the performance and visualize results
print("R^2 is: \n", model.score(X_test, y_test))
predictions = model.predict(X_test)
from sklearn.metrics import mean_squared_error
print('RMSE is: \n', mean_squared_error(y_test, predictions))
```

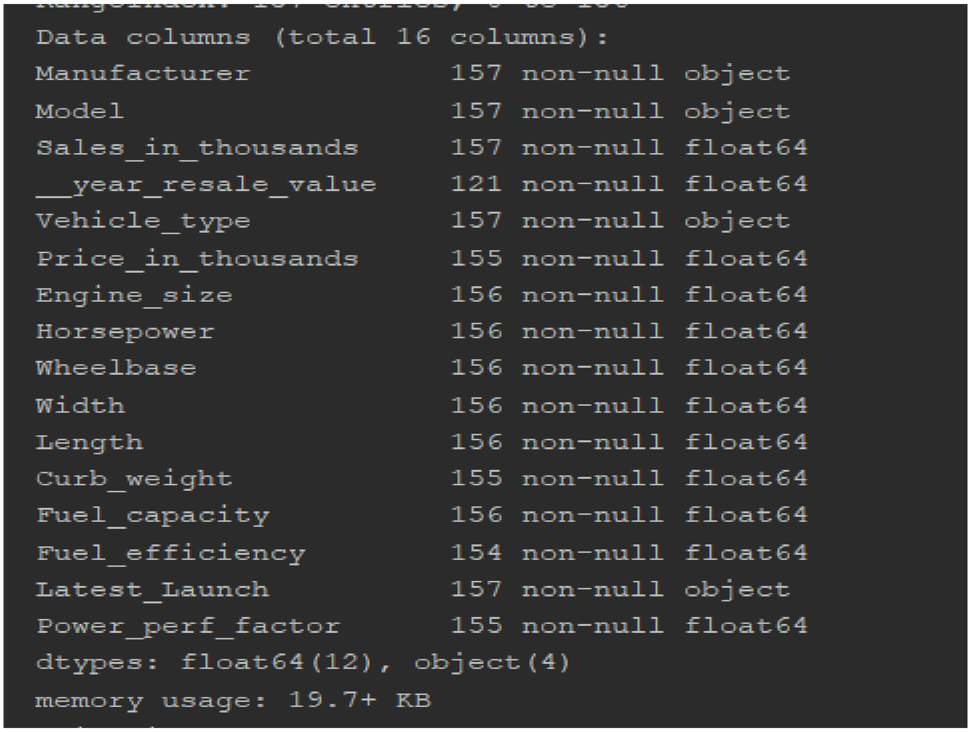
R^2 and RMSE Without EDA will give an error as there are some non-numeric values and also nan values which are not converted as scores can only be defined when they represented in numerical



```
Manufacturer,Model,Sales_in_thousands,__year_resale_value,Vehicle_type,Price_in_thousands,Engine_size,
Acura,Integra,16.919,16.36,Passenger,21.5,1.8,140,101.2,67.3,172.4,2.639,13.2,28,2/2/2012,58.28014952
Acura,TL,39.384,19.875,Passenger,28.4,3.2,225,108.1,70.3,192.9,3.517,17.2,25,6/3/2011,91.37077766
Acura,CL,14.114,18.225,Passenger,,3.2,225,106.9,70.6,192,3.47,17.2,26,1/4/2012,
Acura,RL,8.588,29.725,Passenger,42,3.5,210,114.6,71.4,196.6,3.85,18,22,3/10/2011,91.38977933
Audi,A4,20.397,22.255,Passenger,23.99,1.8,150,102.6,68.2,178,2.998,16.4,27,10/8/2011,62.7776392
Audi,A6,18.78,23.555,Passenger,33.95,2.8,200,108.7,76.1,192,3.561,18.5,22,8/9/2011,84.56510502
Audi,A8,1.38,39,Passenger,62,4.2,310,113,74,198.2,3.902,23.7,21,2/27/2012,134.6568582
BMW,323i,19.747,,Passenger,26.99,2.5,170,107.3,68.4,176,3.179,16.6,26,6/28/2011,71.19120671
BMW,328i,9.231,28.675,Passenger,33.4,2.8,193,107.3,68.5,176,3.197,16.6,24,1/29/2012,81.87706856
BMW,528i,17.527,36.125,Passenger,38.9,2.8,193,111.4,70.9,188,3.472,18.5,25,4/4/2011,83.9987238
Buick,Century,91.561,12.475,Passenger,21.975,3.1,175,109,72.7,194.6,3.368,17.5,25,11/2/2011,71.181451
Buick,Regal,39.35,13.74,Passenger,25.3,3.8,240,109,72.7,196.2,3.543,17.5,23,9/3/2011,95.63670253
Buick,Park Avenue,27.851,20.19,Passenger,31.965,3.8,205,113.8,74.7,206.8,3.778,18.5,24,3/23/2012,85.8
```

Performing EDA:

dataset.info()

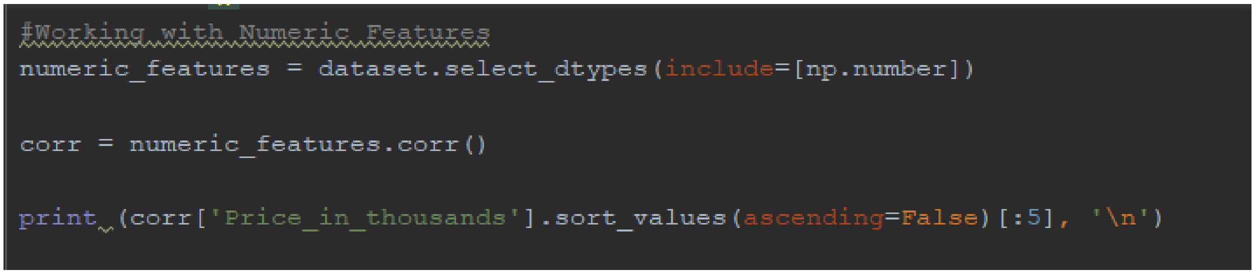


```
Data columns (total 16 columns):
Manufacturer             157 non-null object
Model                    157 non-null object
Sales_in_thousands       157 non-null float64
__year_resale_value      121 non-null float64
Vehicle_type             157 non-null object
Price_in_thousands       155 non-null float64
Engine_size              156 non-null float64
Horsepower               156 non-null float64
Wheelbase                156 non-null float64
Width                    156 non-null float64
Length                   156 non-null float64
Curb_weight              155 non-null float64
Fuel_capacity            156 non-null float64
Fuel_efficiency          154 non-null float64
Latest_Launch            157 non-null object
Power_perf_factor        155 non-null float64
dtypes: float64(12), object(4)
memory usage: 19.7+ KB
```

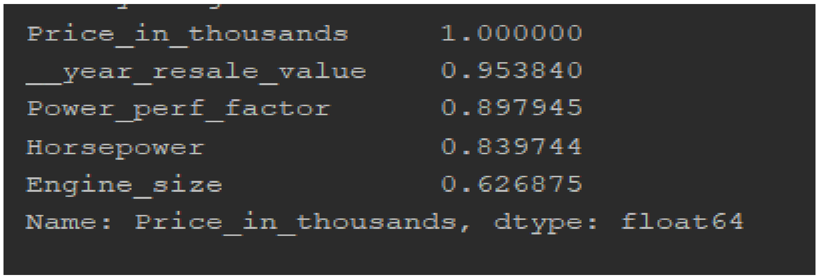
In this dataset, there is no NULL value. We don't need to convert any data into numeric.

We wrote code to sort the columns i.e., features that are most positively correlated and the features that are negatively correlated with the target.

Finding Correlation for better Scores and Good Model:



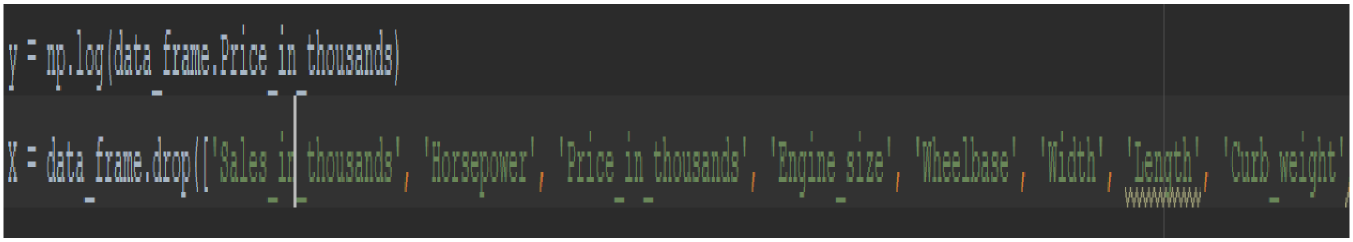
```
#Working with Numeric Features
numeric_features = dataset.select_dtypes(include=[np.number])

corr = numeric_features.corr()

print (corr['Price_in_thousands'].sort_values(ascending=False)[:5], '\n')
```



```
Price_in_thousands      1.000000
__year_resale_value     0.953840
Power_perf_factor       0.897945
Horsepower              0.839744
Engine_size             0.626875
Name: Price_in_thousands, dtype: float64
```

From this result, the feature __year_resale_value , power_per_factor ,Horserpoer

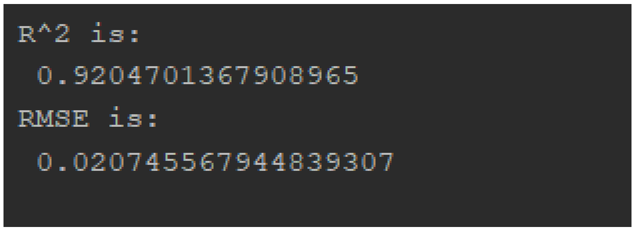
Engine_size should be considered because they have score more than 0.6 and Top 4 Correlated



```
y = np.log(data_frame.Price_in_thousands)
X = data_frame.drop(['Sales_in_thousands', 'Horsepower', 'Price_in_thousands', 'Engine_size', 'Wheelbase', 'Width', 'Length', 'Curb_weight',
```

Without EDA: Error

With EDA:



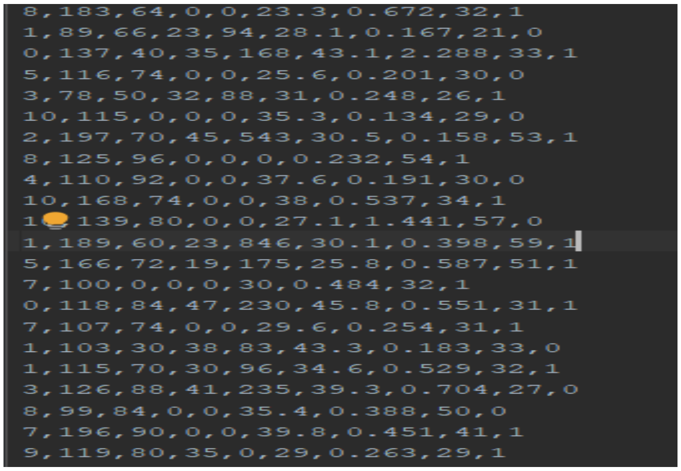
```
R^2 is:
 0.9204701367908965
RMSE is:
 0.020745567944839307
```

Question-5

5. Pick any dataset from the dataset sheet in the class sheet or online which includes both numeric and non-numeric features

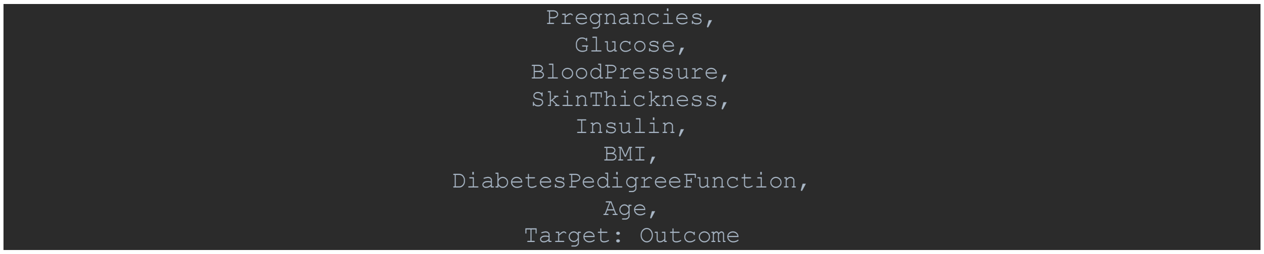
a. Perform exploratory data analysis on the data set (like Handling null values, removing the features not correlated to the target class, encoding the categorical features, ...)

b. Apply the three classification algorithms Naïve Baye's, SVM and KNN on the chosen data set and report which classifier gives better result.



```
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1
5,116,74,0,0,25.6,0.201,30,0
3,78,50,32,88,31,0.248,26,1
10,115,0,0,0,35.3,0.134,29,0
2,197,70,45,543,30.5,0.158,53,1
8,125,96,0,0,0,0.232,54,1
4,110,92,0,0,37.6,0.191,30,0
10,168,74,0,0,38,0.537,34,1
1[illegible]139,80,0,0,27.1,1.441,57,0
1,189,60,23,846,30.1,0.398,59,1
5,166,72,19,175,25.8,0.587,51,1
7,100,0,0,0,30,0.484,32,1
0,118,84,47,230,45.8,0.551,31,1
7,107,74,0,0,29.6,0.254,31,1
1,103,30,38,83,43.3,0.183,33,0
1,115,70,30,96,34.6,0.529,32,1
3,126,88,41,235,39.3,0.704,27,0
8,99,84,0,0,35.4,0.388,50,0
7,196,90,0,0,39.8,0.451,41,1
9,119,80,35,0,29,0.263,29,1
```

If We go to the link mentioned above, we can find the attribute information as follows:

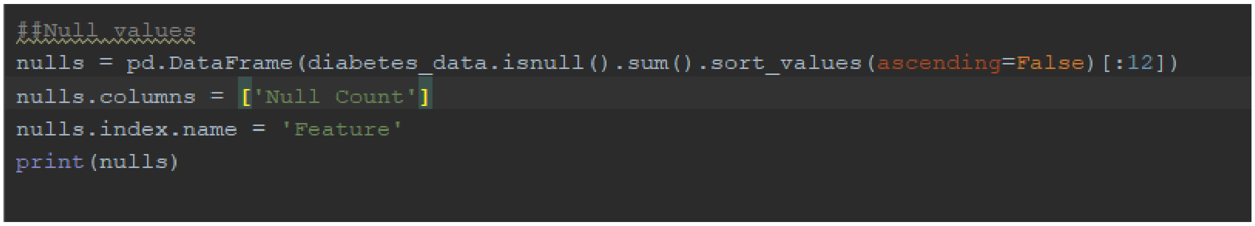


```
Pregnancies,
Glucose,
BloodPressure,
SkinThickness,
Insulin,
BMI,
DiabetesPedigreeFunction,
Age,
Target: Outcome
```

We have six features namely: **Glucose, BloodPressure, Insulin, Age, BMI ,DiabatesPedigreeFunction**

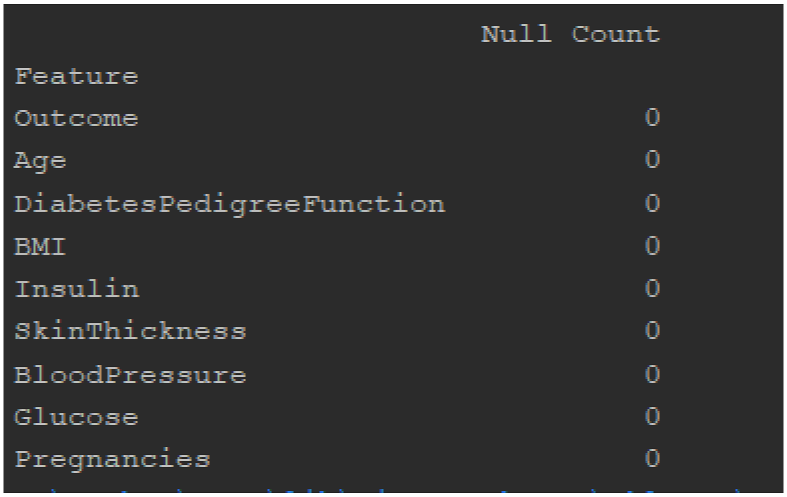
a. Perform EDA

In order to know how many null values present in the dataset, we use these few lines of code



```
##Null values
nulls = pd.DataFrame(diabetes_data.isnull().sum().sort_values(ascending=False)[:12])
nulls.columns = ['Null Count']
nulls.index.name = 'Feature'
print(nulls)
```

The output for the above code will be:

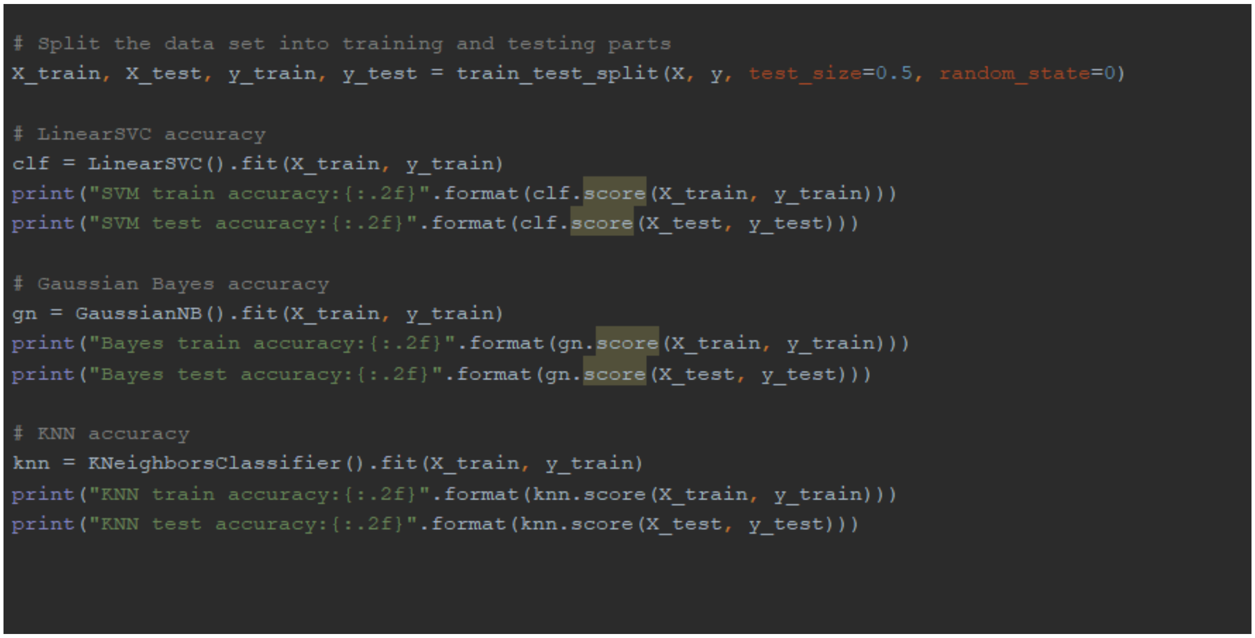


```
                          Null Count
Feature
Outcome                            0
Age                                0
DiabetesPedigreeFunction           0
BMI                                0
Insulin                            0
SkinThickness                      0
BloodPressure                      0
Glucose                            0
Pregnancies                        0
```

So, from the output we can say that there are no **NULL values** in the dataset.

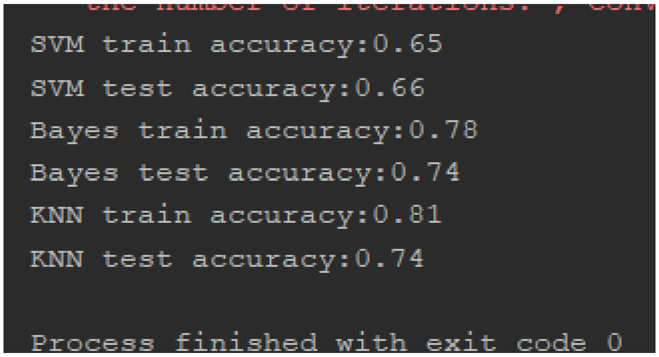
Now, we have to convert the non-numeric features into numeric features by including this piece of code.

By doing this all the non-numeric data will be converted to numeric data. After this we have delete the columns i.e, features which are not corelated to TARGET. For doing this we have to find the correlation of Target with each column.



```
# Split the data set into training and testing parts
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

# LinearSVC accuracy
clf = LinearSVC().fit(X_train, y_train)
print("SVM train accuracy:{:.2f}".format(clf.score(X_train, y_train)))
print("SVM test accuracy:{:.2f}".format(clf.score(X_test, y_test)))

# Gaussian Bayes accuracy
gn = GaussianNB().fit(X_train, y_train)
print("Bayes train accuracy:{:.2f}".format(gn.score(X_train, y_train)))
print("Bayes test accuracy:{:.2f}".format(gn.score(X_test, y_test)))

# KNN accuracy
knn = KNeighborsClassifier().fit(X_train, y_train)
print("KNN train accuracy:{:.2f}".format(knn.score(X_train, y_train)))
print("KNN test accuracy:{:.2f}".format(knn.score(X_test, y_test)))
```

We can see clearly that KNN got high accuracy when compared to GNB.

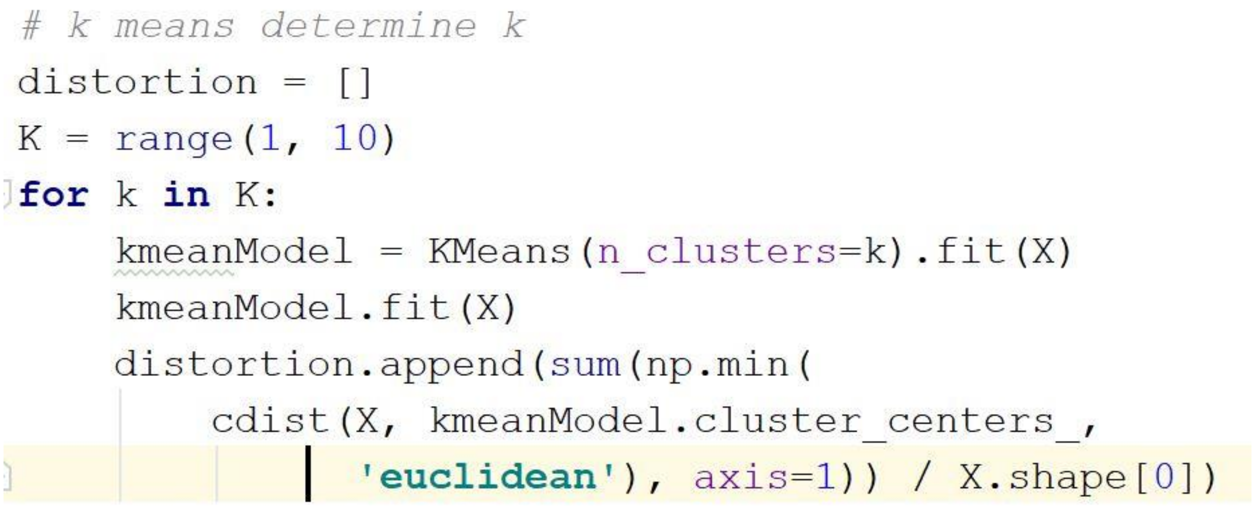


```
    the number of iterations.", conv
SVM train accuracy:0.65
SVM test accuracy:0.66
Bayes train accuracy:0.78
Bayes test accuracy:0.74
KNN train accuracy:0.81
KNN test accuracy:0.74

Process finished with exit code 0
```

Question:6

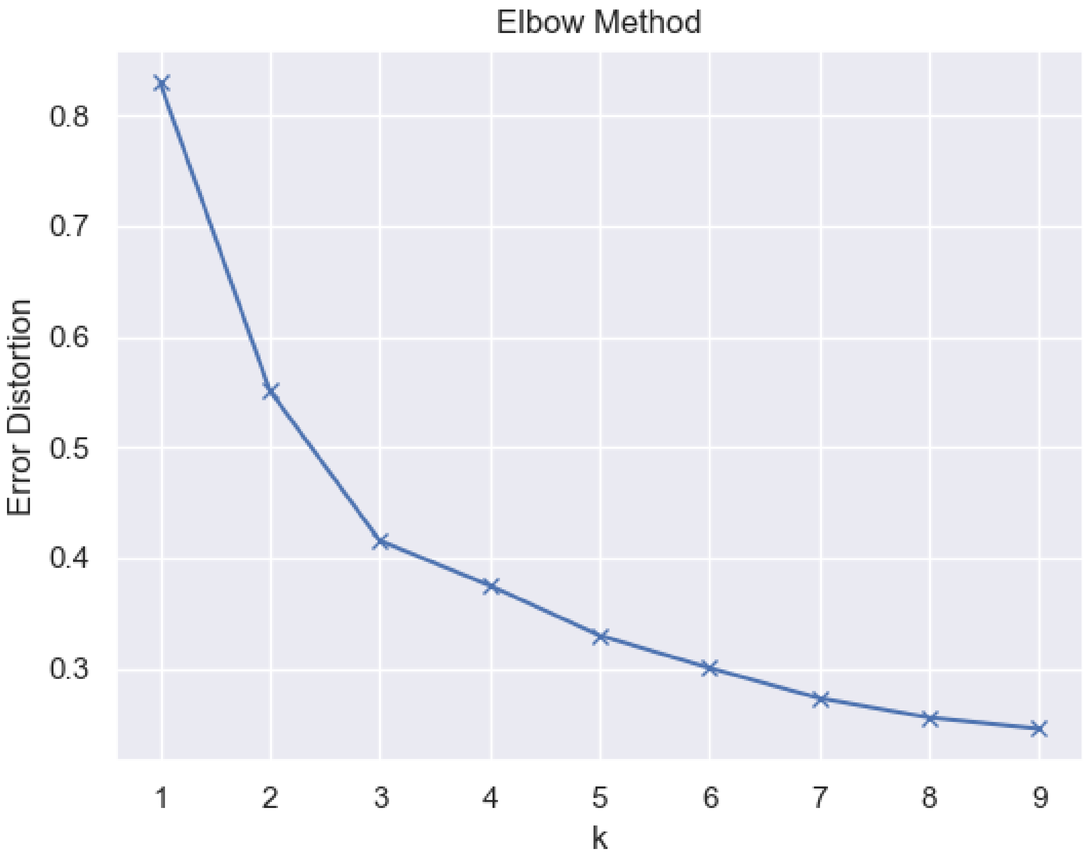
6. Choose any dataset of your choice. Apply K-means on the dataset and visualize the clusters using matplotlib or seaborn.
   a. Report which K is the best using the elbow method.
   b. Evaluate with silhouette score or other scores relevant for unsupervised approaches (before applying clustering clean the data set with the EDA learned in the class)

For this we used iris dataset. Here we used 2 columns named sepal_length and sepal_width.

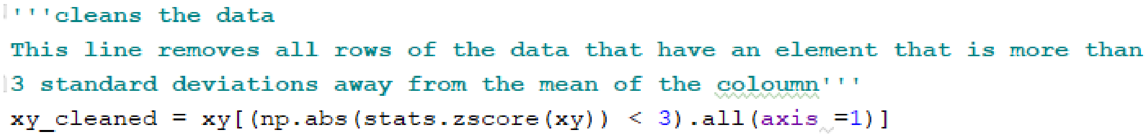


```
# k means determine k
distortion = []
K = range(1, 10)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X)
    kmeanModel.fit(X)
    distortion.append(sum(np.min(
        cdist(X, kmeanModel.cluster_centers_,
              'euclidean'), axis=1)) / X.shape[0])
```

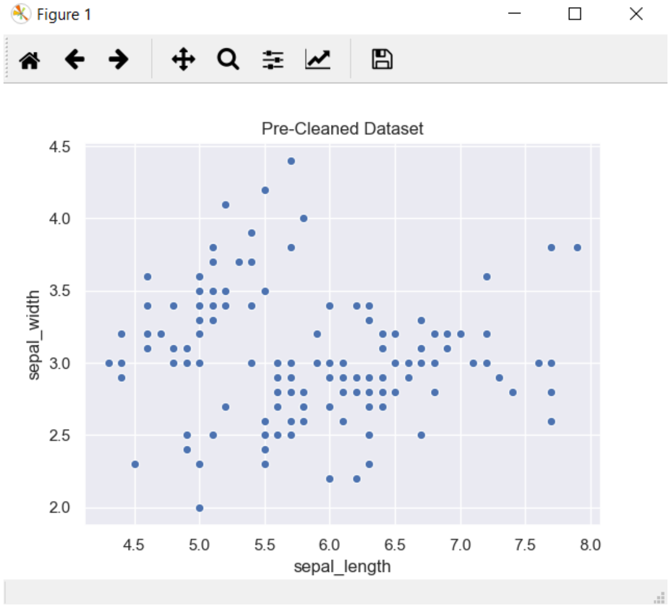
By implementing the above code we can determined the Kmeans is **3.**

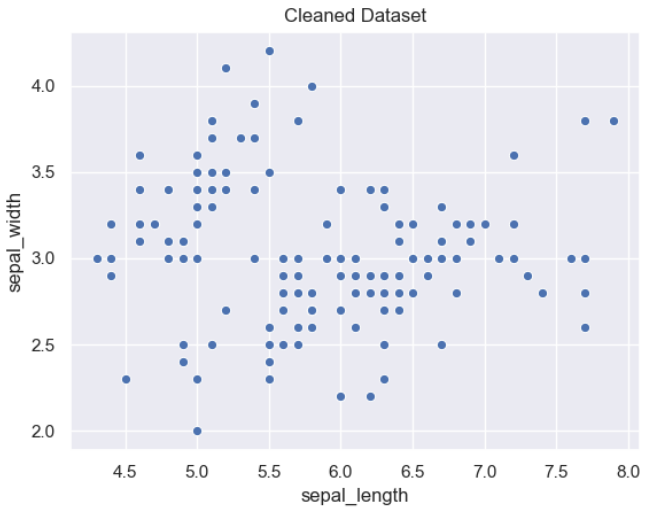


b.

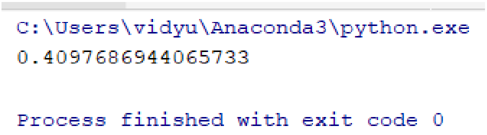


```
'''cleans the data
This line removes all rows of the data that have an element that is more than
3 standard deviations away from the mean of the coloumn'''
xy_cleaned = xy[(np.abs(stats.zscore(xy)) < 3).all(axis =1)]
```





b. Silhouette Score



```
C:\Users\vidyu\Anaconda3\python.exe
0.4097686944065733

Process finished with exit code 0
```