Ashish Pant & Shufeng Li

Class ID: 3 & 20

Python Lab 1

**Introduction**

In this lab assignment we focused on the basic of python and progressed into Machine Learning.

The datasets used in this Assignment are:

1. Iris Species - Classify iris plants into three species in this classic dataset, UCI Machine Learning, <https://www.kaggle.com/uciml/iris>
2. Fish market - Database of common fish species for fish market, Aung Pyae, <https://www.kaggle.com/aungpyaeap/fish-market>
3. Wine Quality, Raj Parmar, <https://www.kaggle.com/rajyellow46/wine-quality>

Submitted by : Ashish Pant(3) and Shufeng Li(20)

**Approaches/Methods**

**Q1. Given a collection of integers that might contain duplicates, nums, return all possible subsets. Do not include null subset.**

Example: Input: [1,2,2]

Output: [ [1], [2], [1,2], [2,2], [1,2,2] ]

**Code**

# define the subsets function to find all possible subsets and do no include subset

def subsets(nums):

    results = []

    # return [] set if the collection is empty

    if not nums:

        return results

    # sort the array to avoid duplicate subsets

    nums.sort()

    length = len(nums)

    # SubsetsHelper function helps us to find subset recursively

    def SubsetsHelper(startIdx, length, subset):

        # check if the subset is not in the results and not a null subset

        if (subset not in results and subset):

            results.append(subset)

        # recursive call SubsetsHelper function

        for i in range(startIdx, length):

            # Increase the start index 1 every time

            SubsetsHelper(i + 1, length, subset + [nums[i]])

    # call the SubsetsHelper function

    SubsetsHelper(0, length, [])

    return results

collection = [1, 2, 2]

subsetsNoNull = subsets(collection)

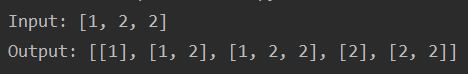
print('Input:', collection)

print('Output:', subsetsNoNull)

**Brief Explanation**

1. Created a function named subset which takes input a list of numbers and returns a set of subsets.
2. First if the list passed is empty it will return empty list.
3. If not empty, then sort the numbers to avoid duplicates.
4. We created a SubsetsHelper function which helps us to find subset recursively.
5. It check if the subset is not in the results and not a null subset
6. Then we do a recursive call to the SubsetsHelper function to find the subset values.

**Output**

****

**Q2. Concatenate two dictionaries and sort the concatenated dictionary by value.**

**Code**

# define 2 dictionaries

x1 = {1: 5, 3: 4, 4: 3, 2: 1, 0: 0}

x2 = {5: 5, 7: 19, 6: 8, 9: 16, 8: 7}

# concat dictionary and sort it using the sorted function

y = {k: v for k, v in sorted({\*\*x1, \*\*x2}.items(), key=lambda item: item[1])}

print(y)

**Brief Explanation**

1. Define 2 example dictionary.
2. I took k(key), v(value) from the list of items that I get when I iterate over the concatenated dictionaries.
3. Storing the sorted dictionary into the variable y

**Output**

****

**Q3. 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.) (Use class, Inheritance, Method Overriding concepts)**

**Code**

from datetime import datetime

from datetime import timedelta

class person():

    def \_\_init\_\_(self, name, age, gender, phoneNumber):

        self.name = name

        self.age = age

        self.gender = gender

        self.phoneNumber = phoneNumber

    # Redefine the print function for person class

    def \_\_str\_\_(self):

        return "\nPerson information:\nName:\t" + self.name + "\tAge:\t" + str(

            self.age) + "\tGender:\t" + self.gender + "\tPhone:\t" + str(

            self.phoneNumber)

# employee class inherits person class

class employee(person):

    def \_\_init\_\_(self, name, age, gender, phoneNumber, employer, departure, arrival, employeeID):

        super(employee, self).\_\_init\_\_(name, age, gender, phoneNumber)

        self.departure = departure

        self.arrival = arrival

        self.employer = employer

        self.employeeID = employeeID

    def \_\_str\_\_(self):

        return "\nEmployee information:\nName:\t" + self.name + "\tAge:\t" + str(

            self.age) + "\tGender:\t" + self.gender + "\tPhone:\t" \

               + str(self.phoneNumber) + "\tdeparture:\t" + str(self.departure) + "\tarrival:\t" + str(self.arrival) \

               + "\temployer:\t" + str(self.employer)

# passenger class inherits person class

class passenger(person):

    def \_\_init\_\_(self, name, age, gender, phoneNumber, departure, arrival):

        super(passenger, self).\_\_init\_\_(name, age, gender, phoneNumber)

        self.departure = departure

        self.arrival = arrival

    def \_\_str\_\_(self):

        return "\nPassenger information:\nName:\t" + self.name + "\tAge:\t" + str(

            self.age) + "\tGender:\t" + self.gender + "\tPhone:\t" \

               + str(self.phoneNumber) + "\tdeparture:\t" + str(self.departure) + "\tarrival:\t" + str(self.arrival)

class flight():

    def \_\_init\_\_(self, flightNum, departure, arrival, flightTime):

        self.flightNum = flightNum

        self.departure = departure

        self.arrival = arrival

        self.flightTime = flightTime

        self.seatsCapacity = 100

        self.seatNum = 0

    def \_\_str\_\_(self):

        return "FlightNum:\t" + self.flightNum + "\tDeparture:\t" + str(self.departure) + "\tarrival:\t" \

               + self.arrival + "\tTime:\t" + self.flightTime

    # Allocate seat for passenger

    def allocateSeatNum(self):

        if self.seatNum < self.seatsCapacity:

            self.seatNum += 1

            print('Your seat number is:', self.seatNum)

    # display the boarding time for passenger

    def getBoardingtime(self):

        flightTime = datetime.strptime(self.flightTime, "%Y-%m-%d %H:%M:%S")

        boardingTime = flightTime + timedelta(hours=-1)

        print('Boarding Time: ', boardingTime)

class airline():

    def \_\_init\_\_(self, airlineName):

        self.airlineName = airlineName

        self.flightNum = 0

        self.employeeNum = 0

        self.flightList = []

        self.employees = []

    # Add employee to the employee list of airline

    def addEmployee(self, employee):

        self.employees.append(employee)

        self.employeeNum += 1

    # add flight to the flight list of airline

    def addFlight(self, flight):

        self.flightList.append(flight)

        self.employeeNum += 1

    # Display the flight information

    def flightsInfor(self):

        for flight in self.flightList:

            print(flight)

    # Book ticket for passenger

    def ticket(self, passenger):

        print('\n====               Welcome to ', self.airlineName, '               ====')

        print('====  Booking ticket from', passenger.departure, ' to', passenger.arrival, '  ====\n')

        for flight in self.flightList:

            if passenger.departure == flight.departure and passenger.arrival == flight.arrival:

                print('Congratulations', passenger.name, '\nYour Flight Number: ', self.airlineName, flight.flightNum)

                flight.allocateSeatNum()

                flight.getBoardingtime()

                break

        print('Sorry: There is no flight available from', passenger.departure, ' to', passenger.arrival, 'now.')

person1 = person("Mike", 25, "Male", 12345678)

print(person1)

employee1 = employee("Jack", 40, "Male", 8165555555, 'Space X', 'MCI', 'SFO', 1)

print(employee1)

passenger1 = passenger("Jack", 40, "Male", 8165555555, 'Kansas City', 'San Francisco')

passenger2 = passenger("Kevin", 20, "Male", 9136666666, 'Kansas City', 'New York')

print(passenger1)

print(passenger2)

flightTime1 = '2017-11-24 17:30:00'

flightTime2 = '2017-11-24 15:00:00'

flightMCItoSFO = flight("1234", "Kansas City", "San Francisco", flightTime1)

flightMCItoLAX = flight("5678", "Kansas City", "Los Angeles", flightTime2)

spaceAir = airline('Space Air')

spaceAir.addFlight(flightMCItoSFO)

spaceAir.addFlight(flightMCItoLAX)

print('\nThe flight of Space Air:')

print(spaceAir.flightsInfor())

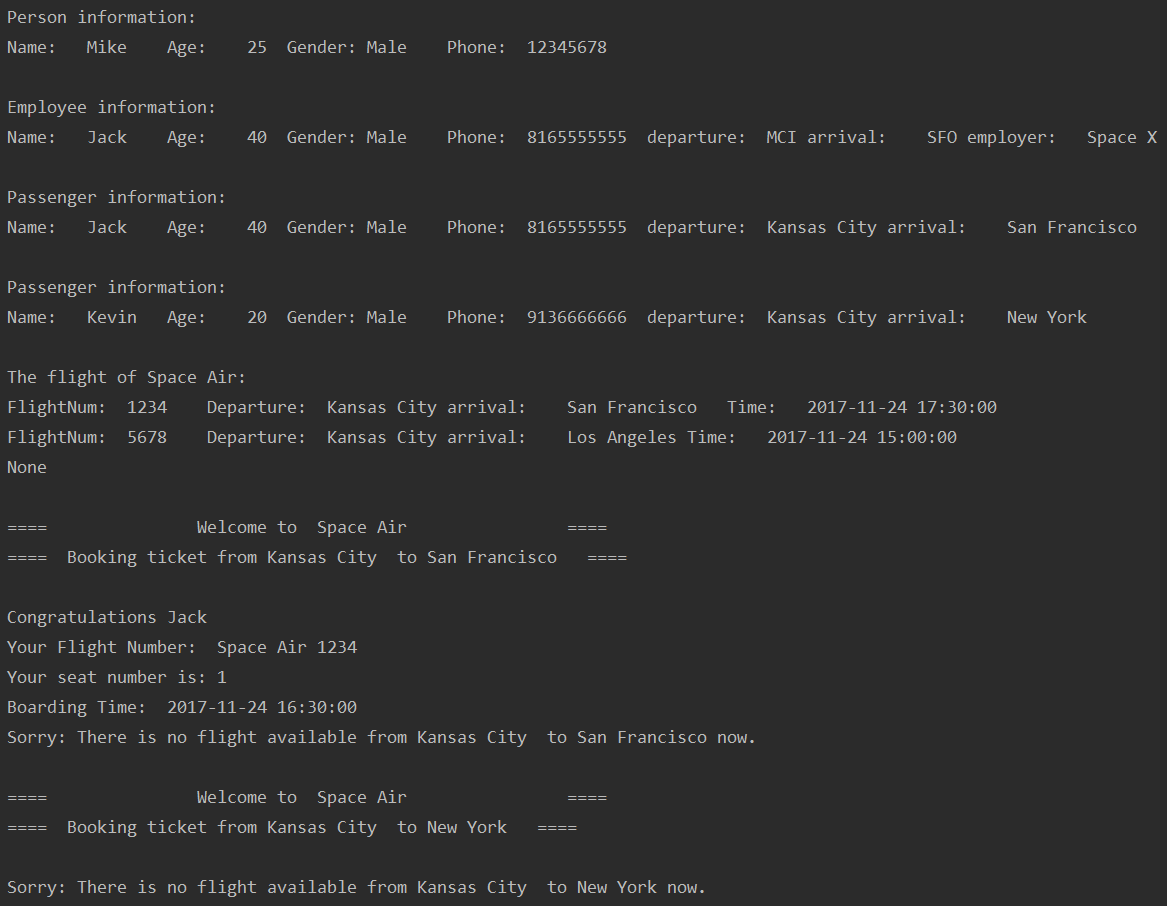
spaceAir.ticket(passenger1)

spaceAir.ticket(passenger2)

**Brief Explanation**

1. Returning the value to print function for person class.
2. Employee class inherits from person class.
3. Passenger class inherits from person class.
4. allocateSeatNum() function used to allocate seat number.
5. getBoardingtime() function to display the boarding time for passenger.
6. addEmployee() function used to add employee to the employee list of airline.
7. addFlight() adds flight to the flight list of airline.
8. flightInfor() displays the flight information.
9. Ticket() Book ticket for passenger.

**Output**

****

**Q4. Go to https://catalog.umkc.edu/course-offerings/graduate/comp-sci/and fetch the course name and overview of course. Hint: Use BeautifulSoup package.**

**Code**

import requests

from bs4 import BeautifulSoup

url = "https://catalog.umkc.edu/course-offerings/graduate/comp-sci"

page = requests.get(url)    # Getting the html page content

soup = BeautifulSoup(page.content, 'html.parser')

# Parsing page content as HTML

# Finding the required information based on the HTML tag and its class

for x, y in zip(soup.find\_all("span", class\_="title"),

soup.find\_all("p", class\_="courseblockdesc")):

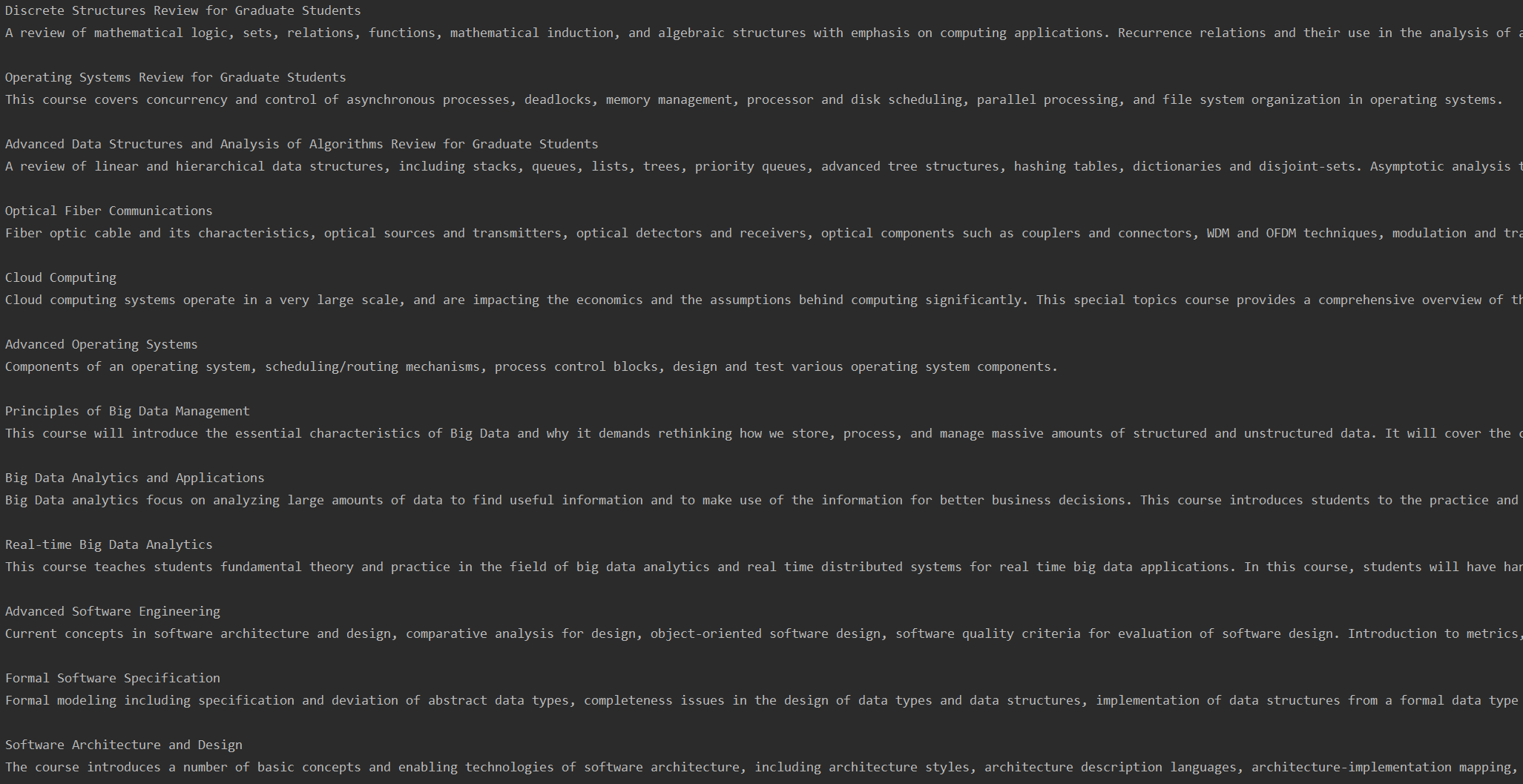
    print(x.string, y.string)

    print()

**Brief Explanation**

1. Using get() function from requests to get the response as a html.
2. Parsing the response as html makes it easy to navigate throught response better.
3. Using find\_all function from beautifulSoup to find the HTML Tag with a class so that required information can be extracted from the page.

**Output**

****

**Q5. 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 andplot different patterns(like Handling null values, removing the features notcorrelated to the target class, encoding the categorical features, ...) b. Apply the three classification algorithms Naïve Bayes, SVM and KNN on the chosen data set and report which classifier gives better result.**

**Code**

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import GaussianNB, MultinomialNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

import warnings

warnings.filterwarnings("ignore")

# Load data fromwinequalityN.csv

wineQuality = pd.read\_csv('winequalityN.csv')

print('The data shape of wineQuality DataFrame:', wineQuality.shape)

print(wineQuality.describe())

# Handling null values

nullValues = pd.DataFrame(wineQuality.isnull().sum().sort\_values(ascending=False))

nullValues.columns = ['Null Number']

nullValues.index.name = 'Feature'

print(nullValues)

print(wineQuality.shape)

# Replace the null values in the wineQuality by the mean

wineQuality = wineQuality.fillna(wineQuality.mean())

nullValues = pd.DataFrame(wineQuality.isnull().sum().sort\_values(ascending=False))

nullValues.columns = ['Null Number']

nullValues.index.name = 'Feature'

print('\nAfter replacing the null values in the wineQuality DataFrame by the mean')

print(nullValues)

print(wineQuality.columns.values)

# see how many samples we have of each species

print(wineQuality["quality"].value\_counts())

print(wineQuality["type"].value\_counts())

numeric\_features = wineQuality.select\_dtypes(include=[np.number])

print('\nThe numeric features: ')

print(numeric\_features.columns.values)

# Encoding a categorical feature

categorical\_feature = wineQuality.select\_dtypes(include='object')

print('\nThe non-numeric features ')

print(categorical\_feature.columns.values)

# ['type'] is a categorical feature

print(wineQuality["type"].value\_counts())

# Converting the value of ['type'] from 'white' to 0, 'red' to 1

type\_mapping = {'white': 0, 'red': 1}

wineQuality['type'] = wineQuality['type'].map(type\_mapping)

print("\nAfter the categorical feature ['type'] :")

print(wineQuality["type"].value\_counts())

features = wineQuality.drop(['quality'], axis=1)

target = wineQuality['quality']

features\_train, features\_test, target\_train, target\_test = \

    train\_test\_split(features, target, test\_size=0.2, random\_state=0)

print('\nThe data shape of features\_train DataFrame:\n', features\_train.shape)

print('\nThe data shape of features\_test DataFrame:\n', features\_test.shape)

# Find the correlation of quality and features

corr = wineQuality.corr()

print('\nThe correlation of quality and features:')

print(corr['quality'].sort\_values(ascending=False), '\n')

# plot a heatmap to show the correlation

plt.subplots()

f1 = plt.figure()

sns.heatmap(corr, cmap="RdYlGn\_r")

plt.show()

f1.savefig("wineQuality.pdf", bbox\_inches='tight')

# plot the relationship between alcohol and quality

f2 = plt.figure()

plt.scatter(wineQuality['alcohol'], wineQuality['quality'], alpha=0.6)

plt.xlabel("alcohol")

plt.ylabel("quality")

plt.show()

f2.savefig("alcohol&quality.pdf", bbox\_inches='tight')

# Naive Bayes model

gnb\_model = GaussianNB()

gnb\_model.fit(features\_train, target\_train)

# Evaluate Naive Bayes model

score\_NaiveBayes = round(gnb\_model.score(features\_train, target\_train), 4)  # ???

print("Naive Bayes score:", "{0:.2%}".format(score\_NaiveBayes))

target\_pred = gnb\_model.predict(features\_test)

print('\nClassification report:')

print(classification\_report(target\_test, target\_pred))

##KNN model

knn = KNeighborsClassifier(n\_neighbors=3)

knn.fit(features\_train, target\_train)

# Evaluate KNN model

score\_knn = round(knn.score(features\_train, target\_train), 4)

print("KNN score:", "{0:.2%}".format(score\_knn))

target\_pred = knn.predict(features\_test)

print('\nClassification report:')

print(classification\_report(target\_test, target\_pred))

# SVM (Support Vector Machine) model

svc = SVC()

svc.fit(features\_train, target\_train)

score\_svc = round(svc.score(features\_train, target\_train), 4)

print("svm accuracy is:", "{0:.2%}".format(score\_svc))

# Evaluate SVM model

target\_pred = svc.predict(features\_test)

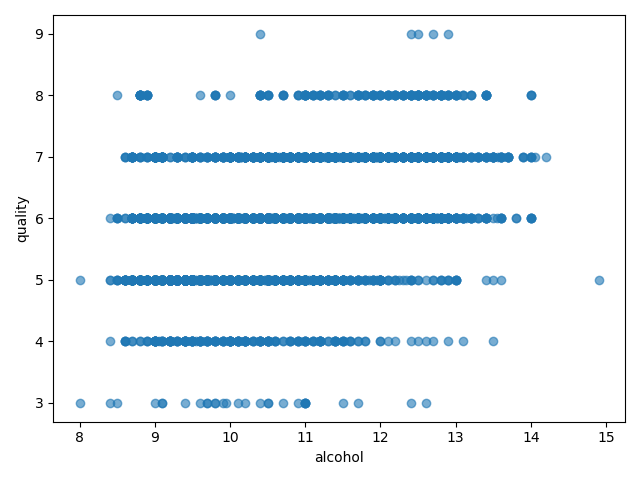
print('\nClassification report:')

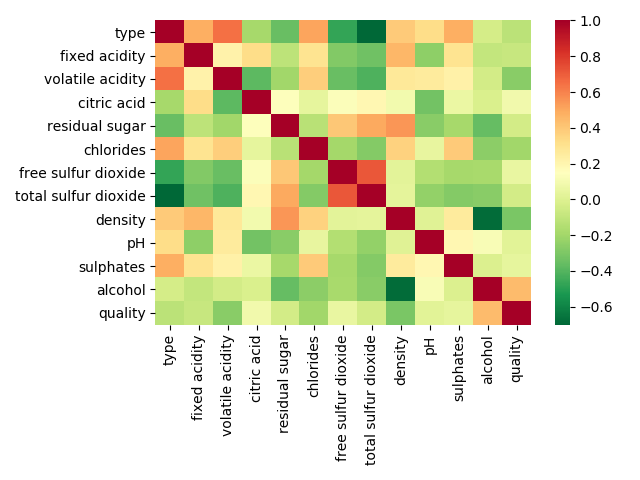
print(classification\_report(target\_test, target\_pred))

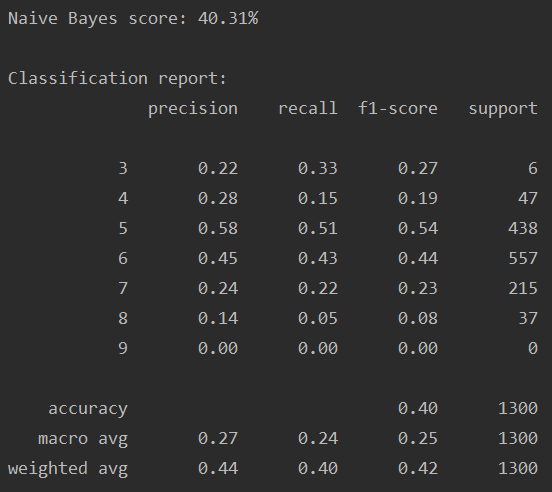
**Brief Explanation**

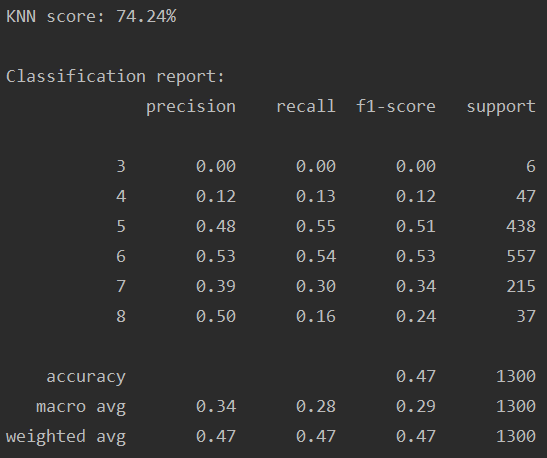
1. Load the data as a CSV file for processing.
2. Finding the null values.
3. Interpolate the values by doing mean.
4. Encoding the data based on the wine name.
5. Find the correlation of quality and features.
6. Using Naïve Bayes, KNN & SVM

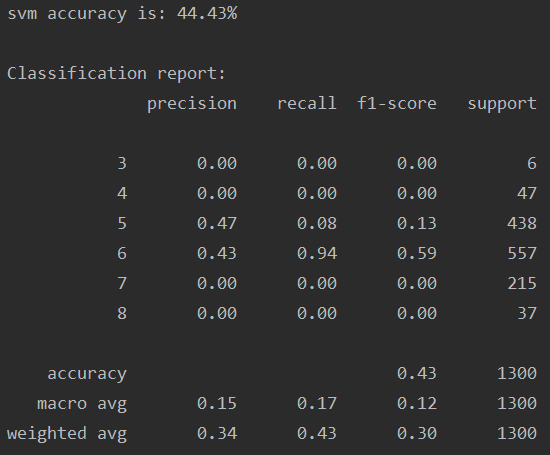
**Output**

****

****

****

****

****

**Q6. Choose any dataset of your choice. Apply K-means on the dataset and visualize the clusters using matplotlib or seaborn.**

1. **Report which K is the best using the elbow method.**
2. **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)**

**Code**

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn import preprocessing

from sklearn.metrics import silhouette\_score

import seaborn as sns

import matplotlib.pyplot as plt

data = pd.read\_csv('Iris\_edit.csv', delimiter=',', usecols=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species'], header=None, skiprows=1, names=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm', 'Species'])

# Null values

nulls = pd.DataFrame(data.isnull().sum().sort\_values(ascending=False)[:25])

nulls.columns = ['Null Count']

nulls.index.name = 'Feature'

print(nulls)

# handling the missing value

data = data.select\_dtypes(include=[np.number]).interpolate().dropna()

#Visualize data in CSV file

sns.FacetGrid(data, hue='Species', height=4).map(plt.scatter, 'SepalLengthCm', 'PetalLengthCm')

plt.show()

sns.FacetGrid(data, hue='Species', height=4).map(plt.scatter, 'SepalWidthCm', 'PetalWidthCm')

plt.show()

sns.FacetGrid(data, hue='Species', height=4).map(plt.scatter, 'SepalWidthCm', 'PetalLengthCm')

plt.show()

# find the top correlated values

numeric\_features = data.select\_dtypes(include=[np.number])

corr = numeric\_features.corr()

print (corr['Species'].sort\_values(ascending=False)[:4], '\n')

# Preprocessing the data using scaler

scaler = preprocessing.StandardScaler()

scaler.fit(data)

X\_scaled\_array = scaler.transform(data)

X\_scaled = pd.DataFrame(X\_scaled\_array, columns = data.columns)

wcss = []

# Finding k using the elbow method

for i in range(2,12):

    kmeans = KMeans(n\_clusters=i,init='k-means++',max\_iter=300,n\_init=10,random\_state=0)

    kmeans.fit(data)

    wcss.append(kmeans.inertia\_)

    score = silhouette\_score(data, kmeans.labels\_, metric='euclidean')

    print("For n\_clusters = {}, silhouette score is {})".format(i, score))

plt.plot(range(1,11),wcss)

plt.title('the elbow method')

plt.xlabel('Number of Clusters')

plt.ylabel('Wcss')

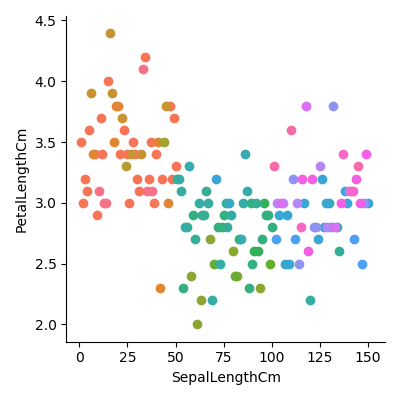
plt.show()

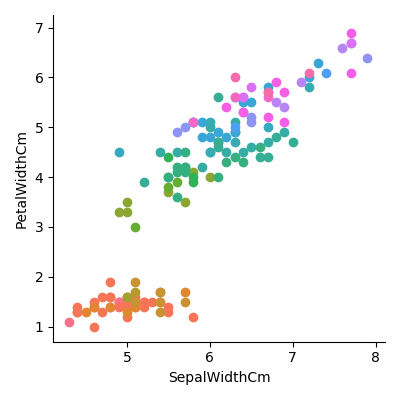
# the best score is shown for nK = 2

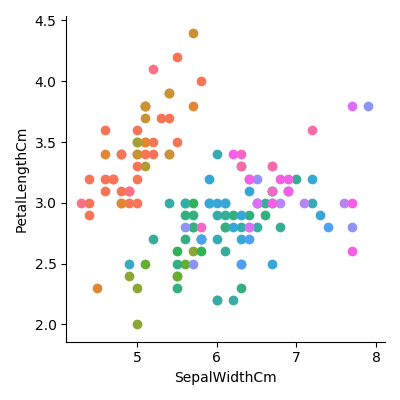
**Brief Explanation**

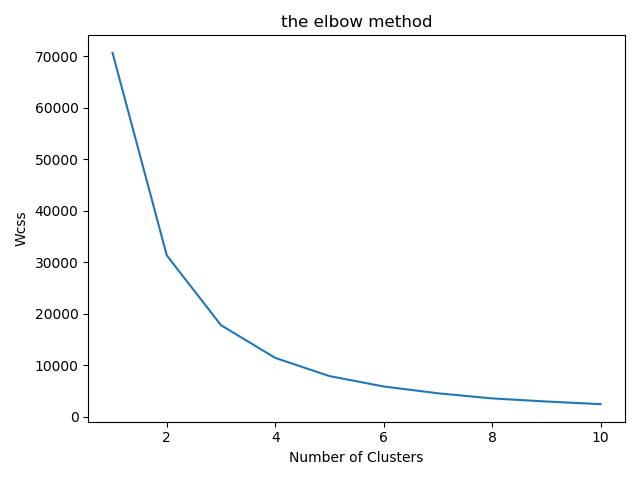
1. First loaded the CSV file and found the null values.
2. Then I interpolated the null values to get fill the null values to get a better result.
3. Visualized the data using graph.
4. Select the most correlated columns.
5. Use scaler so that better model can be trained.
6. Using elbow method found that K = 2 gives the best output.

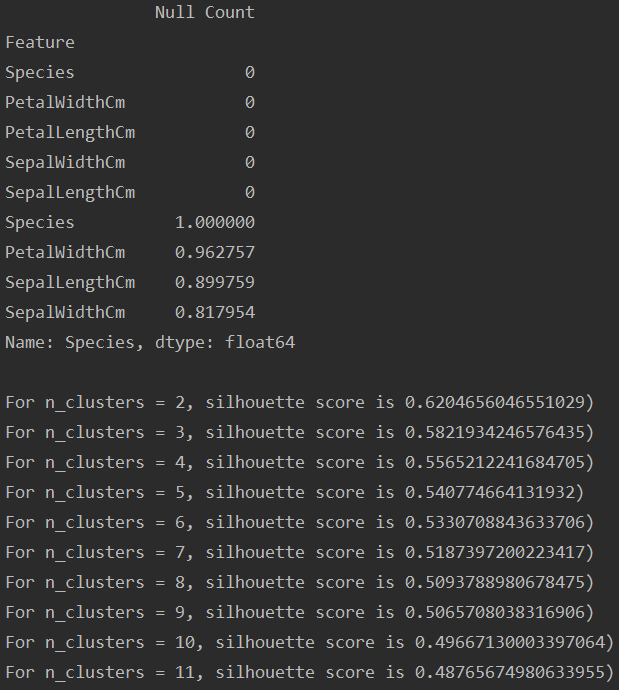
**Output**

****

****

****

****

****

**Q7. Write a program in which take an Input file, use the simple approach below to summarize a text file:**

**Link to input file:** [**https://umkc.box.com/s/7by0f4540cdbdp3pm60h5fxxffefsvrw**](https://umkc.box.com/s/7by0f4540cdbdp3pm60h5fxxffefsvrw)

1. **Read the data from a file.**
2. **Tokenize the text into words and apply lemmatization technique on each word.**
3. **Find all the trigrams for the words.**
4. **Extract the top 10 of the most repeated trigrams based on their count.**
5. **Go through the text in the file.**
6. **Find all the sentences with the most repeated tri-grams.**
7. **Extract those sentences and concatenate.**
8. **Print the concatenated result.**

**Code**

import nltk

from nltk.stem import WordNetLemmatizer

from nltk.util import ngrams

from collections import Counter

# Read the data from nlp\_input.txt and then close the resource.

inputFile = open("nlp\_input.txt", "rb")

originalText = inputFile.read().decode()

inputFile.close()

# Tokenize the words in nlp\_input.txt

wordTokens = nltk.word\_tokenize(originalText)

print("\nWord tokens in nlp\_input.txt:\n", wordTokens)

# Lemmatize all word tokens

lemmatizer = WordNetLemmatizer()

lemmatized\_wordTokens = ', '.join([lemmatizer.lemmatize(wordToken) for wordToken in wordTokens])

print("\nLemmatized word tokens:\n", lemmatized\_wordTokens)

# Find trigrams and save them to a list

n = 3

trigrams = list(ngrams(wordTokens, n))

print("\nAll trigrams:\n", trigrams)

# Count the frequency of trigrams and extract the top 10 repeated trigrams based on the frequency.

top10Trigrams = Counter(trigrams).most\_common(10)

print("\nTop 10 trigrams with frequency:\n", top10Trigrams)

# Remove the frequency and save the top 10 trigrams to a list

top10TrigramsStrs = []

for trigram in top10Trigrams:

    str = ' '.join(trigram[0])

    top10TrigramsStrs.append(str)

print("\nTop 10 trigrams:\n", top10TrigramsStrs)

# Concatenate the sentence with the top 10 trigrams

concatenatedStr = ''

sentTokens = nltk.sent\_tokenize(originalText)

for sToken in sentTokens:

    for top10TrigramsStr in top10TrigramsStrs:

        if (top10TrigramsStr in sToken):

            concatenatedStr += sToken

            break

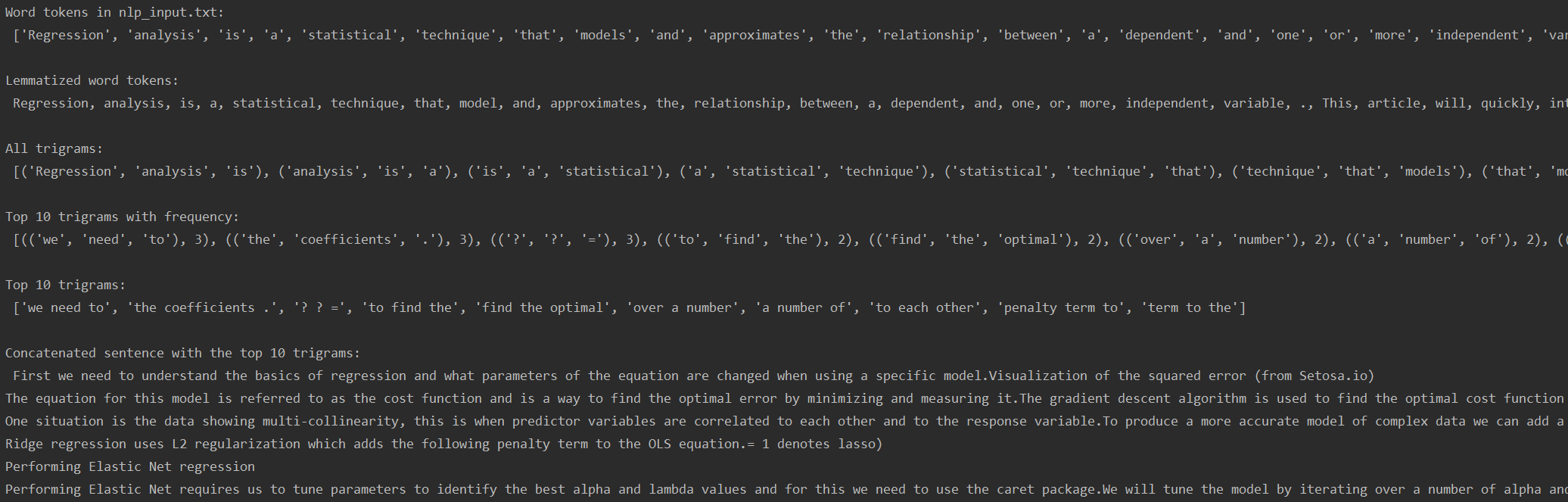
# print the result

print("\nConcatenated sentence with the top 10 trigrams:\n", concatenatedStr)

**Brief Explanation**

1. Read the data from nlp\_input.txt and then close the resource.
2. Tokenize the words in nlp\_input.txt.
3. Lemmatize all word tokens.
4. Find trigrams and save them to a list.
5. Count the frequency of trigrams and extract the top 10 repeated trigrams based on the frequency.
6. Remove the frequency and save the top 10 trigrams to a list.
7. Concatenate the sentence with the top 10 trigrams.
8. Print the result.

**Output**

****

**Q8. 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.**

**Code**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn import linear\_model

import seaborn as sns; sns.set(color\_codes=True)

x = pd.read\_csv('Fish.csv')

# Finding null values

nulls = pd.DataFrame(x.isnull().sum().sort\_values(ascending=False)[:25])

nulls.columns = ['Null Count']

nulls.index.name = 'Feature'

print(nulls)

# Interpolating the null Values

data = x.select\_dtypes(include=[np.number]).interpolate().dropna()

# Plotting correlation for the data

plt.figure(figsize=(20,20))

cor = data.corr()

sns.heatmap(cor, annot=True, cmap="YlGnBu")

plt.show()

# Printing the correlation with target 'Species'

print(cor['Species'].sort\_values(ascending=False)[:5],'\n')

# Build a multiple linear regression model

y = data['Species']

X = data.drop(['Species'],axis =1)

print(X.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42, test\_size=.20)

lr = linear\_model.LinearRegression()

model = lr.fit(X\_train, y\_train)

# 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))

# Visualise the Predicted vs Actual

actual\_values = y\_test

plt.scatter(predictions, actual\_values, alpha=.75, color='r')

# alpha shows overlapping data

plt.xlabel('Predicted ')

plt.ylabel('Actual')

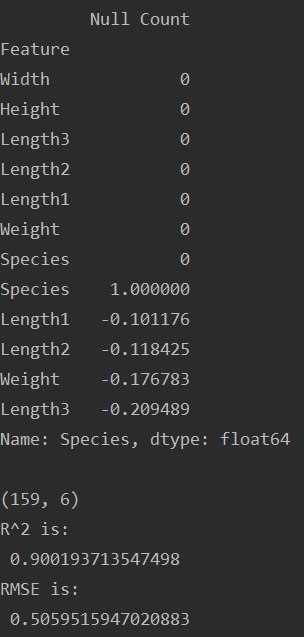
plt.title('Linear Regression Model')

plt.show()

**Brief Explanation**

1. First loaded the CSV file and found the null values.
2. Then I interpolated the null values to get fill the null values to get a better result.
3. Plotted the heatmap for the correlation.
4. Printing correlation value of ‘Species’ will all other columns.
5. Preparing data and then creating the model and then using fit function to train the model.
6. Evaluated the performance of the model using R^2(regression score function) and RMSE(Root Mean Square Error).
7. Created a plot of the predicted vs original values.

**Output**

****

**Conclusion**

We found how to implement different thing in machine learning and find insights from data. By preforming all the tasks we got a better understanding of all the concepts that are explained in the class to us.

**Youtube Link**

https://www.youtube.com/watch?v=QsLp6MMWF0A&list=PLYWbhd6AQk8E0tICz2d6R4LBBmIPGfWnU