Task 1:

from pip.\_vendor.distlib.compat import raw\_input  
  
def SuperSet(arr, n):  
 list = []  
  
 for i in range(2 \*\* n):  
 subset = ""  
  
 for j in range(n):  
  
 if (i & (1 << j)) != 0:  
 subset += str(arr[j]) + "|"  
  
 if subset not in list and len(subset) > 0:  
 list.append(subset)  
  
 for subset in list:  
 arr = subset.split('|')  
 for string in arr:  
 print(string, end=" ")  
 print()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 arr = []  
numbers = int(raw\_input("How many numbers you want to enter?"))  
for i in range(0, numbers):  
 arr.append(int(raw\_input("Enter the number :")))  
n = len(arr)  
SuperSet(arr, n)

Task 2:

n1 = int(input("enter number of items in dictionary 1:"))  
dict1 = {}  
  
for i in range(n1):  
 keys = int(input())  
 values = input()  
 dict1[keys] = values  
print("User input for the dictionary 1:", dict1)  
  
n2 = int(input("enter number of items in dictionary 2:"))  
dict2 = {}  
  
for i in range(n2):  
 keys = int(input())  
 values = input()  
 dict2[keys] = values  
print("User input for the dictionary 2:", dict2)  
  
d = dict(dict1)  
d.update(dict2)  
print("Unsorted:", d)  
ordered = sorted(d.items(), key=lambda x: x[1])  
print("Sorted Dictionary:", ordered)

Task 3:

class Person():  
 def \_\_init\_\_(self, name, email, phone, age):  
 self.name = name  
 self.email = email  
 self.phone = phone  
 self.age = age  
  
 def getname(self):  
 print(" Name:", self.name)  
  
 def getage(self):  
 print("Age:", self.age)  
  
 def getnumber(self):  
 print("Phoneno: " + self.phone)  
  
 def getemail(self):  
 print("Email:", self.email)  
  
  
class FlightInfo():  
 def \_\_init\_\_(self, flight\_number, flight\_name, seat\_number):  
 self.flightnumber = flight\_number  
 self.flightname = flight\_name  
 self.seatnumber = seat\_number  
  
 def getflightname(self):  
 print("Airlines:", self.flightname)  
  
 def getflightnumber(self):  
 print("FlightNo:", self.flightnumber)  
  
 def getseatnumber(self):  
 print("SeatNo:", self.seatnumber)  
  
  
  
class Booking(FlightInfo):  
  
 def \_\_init\_\_(self, source, destination, deptdate,flight\_number,flight\_name,seat\_number,seat\_place):  
 FlightInfo.\_\_init\_\_(self, flight\_number, flight\_name, seat\_number)  
 self.source = source  
 self.destination = destination  
 self.deptdate = deptdate  
 self.flightname = flight\_name  
 self.flightnumber = flight\_number  
 self.seatnumber = seat\_number  
 self.seatplace = seat\_place  
  
 def getsource(self):  
 print("Source :", self.source)  
  
 def getdestination(self):  
 print("Destination :", self.destination)  
  
 def getdeptdate(self):  
 print("Departure Date:", self.deptdate)  
  
 def getseatnumber(self):  
 print("Seat number:", self.seatnumber)  
 print("seat Place", self.seatplace)  
  
  
class Passenger(Person,Booking):  
  
 def \_\_init\_\_(self, name, email, phone, age, flight\_number,  
 flight\_name, seat\_number, deptdate, source, destination,seatplace):  
  
 super().\_\_init\_\_(name, email, phone, age)  
  
 Booking.\_\_init\_\_(self, deptdate, source, destination, flight\_number, flight\_name, seat\_number, seatplace)  
   
 FlightInfo.\_\_init\_\_(self, flight\_number, flight\_name, seat\_number)  
  
  
p = Passenger("Chaitanya", "chaitanya@gmail.com","8166063644", 24, "SPD100", "AirIndia","18", "USA", "India", "10thMarch", "C")  
print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
print("| Passenger Details:")  
print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
p.getname()  
p.getemail()  
p.getage()  
p.getnumber()  
print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
print(" Flight Details ")  
print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
  
p.getflightname()  
p.getflightnumber()  
p.getseatnumber()  
print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
print("| Booking Details:|")  
print(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
p.getsource()  
p.getdestination()  
p.getdeptdate()  
print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  
print("")  
  
f = FlightInfo("Air India", "SPD100", 18)  
f.getseatnumber()

Task 4:

from bs4 import BeautifulSoup  
import requests  
  
URL = 'https://catalog.umkc.edu/course-offerings/graduate/comp-sci/'  
page = requests.get(URL)  
soup = BeautifulSoup(page.content, 'html.parser')  
# print(page.text)  
course\_title = soup.findAll('span', {'class': 'title'})  
course\_overview = soup.findAll('p', {'class': 'courseblockdesc'})  
for text in range(len(course\_title)):  
 print(course\_title[text].text)  
 print(course\_overview[text].text)  
 print('\n')

Task 5:

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import GaussianNB  
  
  
data = pd.read\_csv('corona\_data.csv')  
  
x1 = data.drop('Date', axis=1)  
x2 = x1.drop('Last Update', axis=1)  
x3 = x2.drop('Country', axis=1)  
x4 = x3.drop('Province/State', axis=1)  
  
x4 = x4.drop\_duplicates()  
x4 = x4.dropna()  
x4 = x4.drop("Deaths", axis=1)  
y = x4.isnull().values.any()  
print(y)  
  
  
  
from sklearn.neighbors import KNeighborsClassifier  
  
X\_train = x4.drop("Confirmed", axis=1)  
Y\_train = x4["Confirmed"]  
  
  
X\_test = x4.drop("Recovered",axis=1).copy()  
  
knn = KNeighborsClassifier(n\_neighbors = 3)  
knn.fit(X\_train, Y\_train)  
Y\_pred = knn.predict(X\_test)  
acc\_knn = round(knn.score(X\_train, Y\_train) \* 100, 2)  
print("KNN accuracy is:",acc\_knn)  
  
  
X\_train = x4.drop("Confirmed", axis=1)  
Y\_train = x4["Confirmed"]  
  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_train, Y\_train, test\_size=0.4, random\_state=0)  
  
nav = GaussianNB()  
y\_pred = nav.fit(X\_train, Y\_train).predict(X\_test)  
acc\_knn = round(nav.score(X\_train, Y\_train) \* 100, 2)  
print("NAV accuracy is:", acc\_knn)  
  
from sklearn.svm import SVC  
  
svc = SVC()  
svc.fit(X\_train, Y\_train)  
Y\_pred = svc.predict(X\_test)  
acc\_svc = round(svc.score(X\_train, Y\_train) \* 100, 2)  
print("svm accuracy is:", acc\_svc)

Task 6:

import numpy as np  
import pandas as pd  
  
from sklearn.preprocessing import LabelEncoder  
  
train = pd.read\_csv('winequality-red.csv')  
data = train.select\_dtypes(exclude=[np.number])  
features = list(data.columns)  
le = LabelEncoder()  
  
for i in features:  
 train[i] = le.fit\_transform(train[i])  
data = train.select\_dtypes(include=[np.number]).interpolate().fillna(train.select\_dtypes(include=[np.number]).interpolate().mean(axis=0))  
  
from sklearn import preprocessing  
  
scaler = preprocessing.StandardScaler()  
scaler.fit(data)  
x = scaler.transform(data)  
  
from sklearn.cluster import KMeans  
  
clusters = 3  
seed = 0  
km = KMeans(n\_clusters=clusters, random\_state=seed)  
km.fit(x)  
y\_cluster\_kmeans = km.predict(x)  
  
from sklearn import metrics  
  
score = metrics.silhouette\_score(x, y\_cluster\_kmeans)  
scores = metrics.silhouette\_samples(x, y\_cluster\_kmeans)  
print("Silhoutte score", score)  
  
wcss = []  
for i in range(1, 11):  
 kmeans = KMeans(n\_clusters=i, init='k-means++', max\_iter=500, n\_init=10, random\_state=0)  
 kmeans.fit(data)  
 cluster\_an = kmeans.predict(data)  
 wcss.append(kmeans.inertia\_)  
  
scaler = preprocessing.StandardScaler()  
scaler.fit(x)  
x\_scaled\_array = scaler.transform(x)  
x\_scaled = pd.DataFrame(x\_scaled\_array)  
feature\_scaling\_score = metrics.silhouette\_score(x\_scaled, y\_cluster\_kmeans)  
print("Feature scaling score:", feature\_scaling\_score)  
  
from sklearn.decomposition import PCA  
  
pca = PCA(2)  
x\_pca = pca.fit\_transform(x\_scaled)  
  
km\_1 = KMeans(n\_clusters=3, random\_state=0)  
km\_1.fit(x\_pca)  
kmeans\_1 = km\_1.predict(x\_pca)  
  
y\_cluster\_kmeans\_1 = km\_1.predict(x\_pca)  
  
pca\_score = metrics.silhouette\_score(x\_pca, y\_cluster\_kmeans\_1)  
print("Pca Score:",pca\_score)  
  
  
  
import matplotlib.pyplot as plt  
  
plt.plot(range(1, 11), wcss)  
plt.title('the elbow method')  
plt.xlabel('Number of Clusters')  
plt.ylabel('Wcss')  
plt.show()  
  
plt.scatter(y\_cluster\_kmeans, scores, alpha=.75,color='b')  
plt.xlabel('Cluster')  
plt.ylabel('Scores')  
plt.show()  
  
plt.scatter(range(1, len(y\_cluster\_kmeans) + 1), y\_cluster\_kmeans, alpha=.75,color='b')  
plt.xlabel('Id')  
plt.ylabel('Cluster')  
plt.show()  
  
plt.hist(y\_cluster\_kmeans, color="blue")  
plt.xlabel('Type')  
plt.ylabel('Count')  
plt.title('K-Means Model')  
plt.show()

Task 7:

file = open("nlp\_input.txt", "r")  
string = file.read()  
import nltk  
  
nltk.download('punkt')  
nltk.download('averaged\_perceptron\_tagger')  
nltk.download('wordnet')  
nltk.download('maxent\_ne\_chunker')  
nltk.download('words')  
  
stokens = nltk.sent\_tokenize(string)  
wtokens = nltk.word\_tokenize(string)  
  
tagged = nltk.pos\_tag(wtokens)  
for k in tagged:  
 print(k)  
  
from nltk.stem import WordNetLemmatizer  
  
lemmatizer = WordNetLemmatizer()  
for t in wtokens[:10]:  
 print("Lemmatizer:", lemmatizer.lemmatize(t), ", With POS=n:", lemmatizer.lemmatize(t, pos="n"))  
  
from nltk.util import ngrams  
  
token = nltk.word\_tokenize(string)  
  
for s in stokens[:10]:  
 trigrams = list(ngrams(token, 3))  
 print("The text:", s, "\ntrigrams", trigrams)  
  
wordfreq = nltk.FreqDist(trigrams)  
mostcommon = wordfreq.most\_common(10)  
print("Most frequently repeated top 10 trigrams:", mostcommon)  
  
sentTokens = nltk.sent\_tokenize(string)  
concatenatedArray = []  
for sentence in sentTokens:  
 for (i, j, k) in trigrams:  
 for ((l, m, n), length) in mostcommon:  
 if (i,j,k == l, m, n):  
 concatenatedArray.append(sentence)  
  
print("\nConcatenated sentenced is:", concatenatedArray)

Task 8:

import pandas as pd  
from sklearn import linear\_model  
from sklearn.metrics import r2\_score, mean\_squared\_error  
  
data = pd.read\_csv('corona\_data.csv')  
x1 = data.drop('Date', axis=1)  
x2 = x1.drop('Last Update',axis=1)  
x3 = x2.drop('Country',axis=1)  
x4 = x3.drop('Province/State', axis=1)  
  
y = x4.isnull().values.any()  
print(y)  
x4 = x4.drop\_duplicates()  
x4 = x4.dropna()  
  
X\_train = x4.drop('Confirmed', axis=1)  
Y\_train = x4['Confirmed']  
  
regr = linear\_model.LinearRegression()  
  
regr.fit(X\_train, Y\_train)  
y\_pred = regr.predict(X\_train)  
  
print("Variance score: %.2f" % r2\_score(Y\_train,y\_pred))  
print("Mean squared error: %.2f" % mean\_squared\_error(Y\_train,y\_pred))