1.py

**def** TuptoDict(tupe, d):  
 **for** a, b **in** tupe:  
 d.setdefault(a, []).append(b)  
 **return** d  
  
tup\_input = [( **"John"**, (**"Physics"**, 80)), (**"Daniel"**, (**"Science"**, 90)),  
(**"John"**, (**"Science"**, 95)), (**"Mark"**,(**"Maths"**, 100)), (**"Daniel"**, (**"History"**, 75)), (**"Mark"**, (**"Social"**, 95))]  
  
dict1 = {}  
dict2 = {}  
  
dict2 = TuptoDict(tup\_input, dict1)  
print(**"output before sorting : "**, dict2)  
  
dict\_values = list(dict2.values())  
  
list1 = []  
  
*# soring values based on subject names***for** i **in** dict\_values:  
 sorted\_list = sorted(i, key=**lambda** item: item[0],reverse=**False**)  
 list1.append(sorted\_list)  
  
k=0  
*# reassigning sorted values to the keys of dictionary***for** key **in** dict2:  
 dict2[key] = list1[k]  
 k=k+1  
print(**"final output after sorting : "**, dict2)

2.py

**def** long\_substrs(str1):  
 curr\_list = [] *#for holding current string* sub\_string\_list = [] *#for holding all sub strings  
  
# for generating list of subtrings without repeating characters* **for** letter **in** str1:  
 **if** letter **in** curr\_list:  
 sub\_string\_list.append(**''**.join(curr\_list))  
 curr\_pos = curr\_list.index(letter) + 1  
 curr\_list = curr\_list[curr\_pos:]  
 *# print(curr\_list)* curr\_list += letter  
 sub\_string\_list.append(**''**.join(curr\_list))  
 *# print(sub\_string\_list)  
  
# for displaying only substrings that has maximum length of all substrings* max\_len = max(len(k) **for** k **in** sub\_string\_list)  
 **for** k **in** sub\_string\_list:  
 **if** len(k) == max\_len:  
 print((k, len(k)))  
  
str = input(**"Enter a String : "**)  
long\_substrs(str)

3.py

**class** Flight:  
 **def** \_\_init\_\_(self, f\_name, f\_num):  
 self.f\_name = f\_name  
 self.f\_num = f\_num  
  
 **def** flight\_display(self):  
 print(**'flight\_name: '**, self.f\_name)  
 print(**'flight\_number: '**, self.f\_num)  
  
**class** Employee:  
 **def** \_\_init\_\_(self, e\_id, e\_name, e\_age, e\_gender):  
 self.e\_name = e\_name  
 self.e\_age = e\_age  
 self.\_\_e\_id = e\_id *#private variable* self.e\_gender = e\_gender  
 **def** emp\_display(self):  
 print(**"employee\_name: "**,self.e\_name)  
 print(**'employee\_id: '**, self.\_\_e\_id)  
 print(**'employee\_age: '**,self.e\_age)  
 print(**'employee\_gender: '**, self.e\_gender)  
  
**class** Passenger:  
 **def** \_\_init\_\_(self):  
 Passenger.\_\_passport\_number = input(**"enter the passport number of passenger: "**) *#private variable* Passenger.name = input(**'enter name of passenger: '**)  
 Passenger.age = int(input(**'enter age of passenger: '**))  
 Passenger.gender = input(**'enter the gender: '**)  
 Passenger.class\_type = input(**'select b for business or e for economy class: '**)  
  
**class** Luggage():  
 bag\_fare = 0  
 **def** \_\_init\_\_(self, checkin\_bags):  
 self.checkin\_bags = checkin\_bags  
 **if** checkin\_bags > 2 :  
 self.bag\_fare = (checkin\_bags-2)\*40  
 **else**:  
 **pass  
  
class** Fare(Luggage): *#inheritance* offline = 250  
 online = 300  
 total\_fare=0  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_(int(input(**'enter number of check-in bags carrying other than cabin bag : '**))) *#super call* x = input(**'buy ticket through online or offline: '**)  
 **if** x == **'online'**:  
 Fare.total\_fare = self.online + self.bag\_fare  
 **elif** x == **'offline'**:  
 Fare.total\_fare = self.offline + self.bag\_fare  
 **else**:  
 **pass  
  
class** Ticket(Passenger, Fare): *#inheritance* **def** \_\_init\_\_(self):  
 print(**"Passenger name:"**,Passenger.name)  
 **if** Passenger.class\_type == **"b"**:  
 str = **"business"** Fare.total\_fare+=80  
 **else**:  
 str = **"economy"  
 pass** print(**"Passenger class type:"**,str)  
 print(**"Total fare:"**,Fare.total\_fare)  
  
  
f1=Flight(**'AA'**,**'AA786J'**)  
f1.flight\_display()  
  
emp1 = Employee(**'e1'**,**'Gopi Chand'**,24,**'M'**)  
emp1.emp\_display()  
  
p1 = Passenger()  
  
fare1=Fare()  
  
t= Ticket()

4.py

**import** pandas **as** pd  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
advv = pd.read\_csv(**'advertising.csv'**)  
  
print(advv.columns)  
  
*#converting non-numeric features to numeric features*advv[**'Country'**] =advv[**'Country'**].astype(**'category'**).cat.codes  
advv[**'Ad Topic Line'**] =advv[**'Ad Topic Line'**].astype(**'category'**).cat.codes  
advv[**'City'**] =advv[**'City'**].astype(**'category'**).cat.codes  
  
*#checking for nulls in the data*nulls = pd.DataFrame(advv.isnull().sum().sort\_values(ascending=**False**))  
nulls.columns = [**'Null Count'**]  
nulls.index.name = **'Feature'***# print(nulls)  
  
#removing nulls in data*adv = advv.select\_dtypes(include=[np.number]).interpolate().dropna()  
print(sum(adv.isnull().sum() != 0))  
  
  
print(adv.head())  
  
df = pd.DataFrame(adv)  
  
*# checking correlation of all the columns against target column*print(**' correlation of all columns are :\n'** + str(df[df.columns[:]].corr()[**'Clicked on Ad'**].sort\_values(ascending=**False**)))  
  
*# Taking all columns for analysis*X = adv.drop(**'Clicked on Ad'**,axis=1)  
y = adv[**'Clicked on Ad'**]  
  
*# taking top correlated columns for analysis*A = adv[[**'Age'**,**'Ad Topic Line'**,**'Country'**]]  
b = adv[**'Clicked on Ad'**]  
  
*# splitting data into train data for training model and test data for testing model***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=.2)  
A\_train, A\_test, b\_train, b\_test = train\_test\_split(A, b, random\_state=42, test\_size=.2)  
  
*# fitting model to our train data***from** sklearn.linear\_model **import** LinearRegression  
lm = LinearRegression()  
lm.fit(X\_train,y\_train)  
  
lm2 = LinearRegression()  
lm2.fit(A\_train,b\_train)  
  
*# print(lm.intercept\_)*coeff = pd.DataFrame(lm.coef\_,X.columns,columns=[**'Coefficient'**])  
*# print(coeff)  
  
# testing the model comparing with test data*predictions = lm.predict(X\_test)  
predictions1 = lm2.predict(A\_test)  
plt.scatter(y\_test,predictions)  
*# plt.show()  
  
# metrics***from** sklearn.metrics **import** r2\_score  
print(**'total r2 score is '**,r2\_score(y\_test,predictions))  
print(**'correlated r2 score is '**,r2\_score(b\_test,predictions1))  
*# print('r2 score is ',lm.score(X\_test,y\_test))  
# print('r2 score is ',lm.score(y\_test,predictions))***from** sklearn.metrics **import** mean\_squared\_error  
print(**'total rmse'**,mean\_squared\_error(y\_test,predictions))  
print(**'correlated rmse'**,mean\_squared\_error(b\_test,predictions1))

5.py

**from** sklearn.preprocessing **import** LabelEncoder  
  
le=LabelEncoder()  
**import** pandas **as** pd  
**import** numpy **as** np  
df = pd.read\_csv(**'Loan payments data.csv'**)  
print(df.info())  
  
*#converting non-numeric features to numeric features using label encoder*df[**'loan\_status'**]=le.fit\_transform(df[**'loan\_status'**])  
df[**'Gender'**]=le.fit\_transform(df[**'Gender'**])  
df[**'education'**]=le.fit\_transform(df[**'education'**])  
df[**'past\_due\_days'**]=le.fit\_transform(df[**'past\_due\_days'**])  
  
print(df.head())  
*#checking for nulls in the data*nulls = pd.DataFrame(df.isnull().sum().sort\_values(ascending=**False**))  
nulls.columns = [**'Null Count'**]  
nulls.index.name = **'Feature'***# print(nulls)  
  
#removing nulls in data*df1 = df.select\_dtypes(include=[np.number]).interpolate().dropna()  
print(sum(df1.isnull().sum() != 0))  
  
*# checking correlation of all the columns against target column*print(**' correlation of all columns are :\n'** + str(df1[df1.columns[:]].corr()[**'loan\_status'**].sort\_values(ascending=**False**)))  
  
*# taking top correlated columns for analysis*A = df1[[**'past\_due\_days'**,**'age'**,**'education'**]]  
*# taking target column*b = df1[**'loan\_status'**]  
  
*# splitting data into train data for training model and test data for testing model***from** sklearn.model\_selection **import** train\_test\_split  
A\_train, A\_test, b\_train, b\_test = train\_test\_split(A, b, random\_state=42, test\_size=.1)  
  
*# using naive bayes model***from** sklearn.naive\_bayes **import** GaussianNB  
gnb = GaussianNB()  
  
*#using svm***from** sklearn.svm **import** SVC  
model = SVC()  
  
*#using knn***from** sklearn.neighbors **import** KNeighborsClassifier  
knn = KNeighborsClassifier()  
  
  
*# fitting model to our train data*gnb.fit(A\_train,b\_train)  
model.fit(A\_train,b\_train)  
knn.fit(A\_train,b\_train)  
  
*# testing the model comparing with test data*predictions\_naive = gnb.predict(A\_test)  
predictions\_svm = model.predict(A\_test)  
predictions\_knn = knn.predict(A\_test)  
  
  
*# calculating root mean sqaure value of the errors***from** sklearn.metrics **import** mean\_squared\_error  
print (**'RMSE for naive bayes is: \n'**, mean\_squared\_error(b\_test, predictions\_naive))  
print (**'RMSE for svm is: \n'**, mean\_squared\_error(b\_test, predictions\_svm))  
print (**'RMSE for knn is: \n'**, mean\_squared\_error(b\_test, predictions\_knn))  
  
  
**from** sklearn **import** metrics  
print(**"Accuracy for naive bayes is : "**,round(metrics.accuracy\_score(b\_test, predictions\_naive) \* 100, 2))

print(**"Accuracy for svm is : "**,round(metrics.accuracy\_score(b\_test, predictions\_svm) \* 100, 2))

print(**"Accuracy for knn is : "**,round(metrics.accuracy\_score(b\_test, predictions\_knn) \* 100, 2))

6.py

**import** pandas **as** pd  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.decomposition **import** PCA  
**import** seaborn **as** sns  
**from** scipy.spatial.distance **import** cdist  
  
data = pd.read\_csv(**'USA\_Housing.csv'**)  
house = data.select\_dtypes(include=[np.number]).interpolate().dropna()  
print(sum(house.isnull().sum() != 0))  
nulls = pd.DataFrame(house.isnull().sum().sort\_values(ascending=**False**))  
nulls.columns = [**'Null Count'**]  
nulls.index.name = **'Feature'**print(nulls)  
  
print(**' correlation of all columns are :\n'** + str(data[data.columns[:]].corr()[**'Price'**].sort\_values(ascending=**False**)))  
  
sns.FacetGrid(data,size=7)\  
.map(plt.scatter,**'Avg. Area Number of Rooms'**,**'Price'**)\  
.add\_legend()  
plt.show()  
  
x = house.iloc[:,[0,1,2,3,4,5]]  
y = house.iloc[:,-1]  
print(x.shape, y.shape)  
  
  
**from** sklearn **import** preprocessing  
  
scaler = preprocessing.StandardScaler()  
**from** sklearn **import** metrics  
scaler.fit(x)  
X\_scaled\_array = scaler.transform(x)  
X\_scaled = pd.DataFrame(X\_scaled\_array, columns = x.columns)  
  
  
**from** sklearn.cluster **import** KMeans  
Sum\_of\_squared\_distances = []  
K = range(1,11)  
**for** k **in** K:  
 km = KMeans(n\_clusters=k,init=**'k-means++'**,max\_iter=300,n\_init=10,random\_state=0)  
 km = km.fit(X\_scaled)  
 Sum\_of\_squared\_distances.append(km.inertia\_)  
  
  
plt.plot(K, Sum\_of\_squared\_distances, **'bx-'**)  
plt.xlabel(**'k'**)  
plt.ylabel(**'Sum\_of\_squared\_distances'**)  
plt.title(**'The Elbow Method showing the optimal k'**)  
plt.show()  
  
  
nclusters = 3 *# this is the k in kmeans*seed = 0  
K = range(2,10)  
**for** k **in** K:  
 km = KMeans(n\_clusters=k, random\_state=seed)  
 km.fit(X\_scaled)  
*# predict the cluster for each data point* y\_cluster\_kmeans = km.predict(X\_scaled)  
 plt.scatter(X\_scaled\_array[:, 2], X\_scaled\_array[:, 5], c=y\_cluster\_kmeans, s=50)  
 centers = km.cluster\_centers\_  
 plt.scatter(centers[:, 2], centers[:, 5], c=**'black'**, s=200, alpha=0.5)  
 plt.show()  
 score = metrics.silhouette\_score(X\_scaled, y\_cluster\_kmeans)  
 print(**'silhouette score for clusters '** +str(k)+**' is : '** + str(score))