SOURCE CODE

1st_Solution:

```
# importing the itertools library

import itertools
# creating an empty list
Elements_list = []
# number of elements as input
n = int(input("Enter the number of elements of array: "))
print("Enter the values for elements:")
# iterating till the range
for i in range(0, n):
    elements = int((input()))
# adding the elements to the Elements_list
    Elements_list.append(elements)
for i in range(1,len(Elements_list)+1):
# Prints all possible combinations of r elements in a given array of size n
   a = (list(itertools.combinations(Elements_list,i)))

# To eliminate the duplicate combinations in the subsets
   print(list(set(a)))
```

2nd_Solution:

```
# Create first dictionary
dict1 = {'Vishnu': 5, 'Vandith': 7, 'Jeswanth': 10}
# Create second dictionary
dict2 = {'Haneesh': 8, 'Vishnu': 20, 'Harun': 11}
# Merge contents of dict2 in dict1
dict1.update(dict2)
# Printing the content of the dictionary1
print('Updated dictionary 1 :')
print(dict1)
# Create a list of tuples sorted by index 1 i.e. value field
listofTuples = sorted(dict1.items(), key=lambda x: x[1])
# Iterate over the sorted sequence
for elem in listofTuples:
    print(elem[0], " ::", elem[1])
```

3rd solution:

```
# Airline class which takes input from the terminal and generates flight number.
class Airline:
    def __init__(self, flightNo):
        Airline.source = input('Enter Src : ')
        Airline.destination = input('Enter Dest : ')
        print('Airlines that are available :')
        print('DELTA')
        print('SPICEJET')
        print('INDIGO')
        print('QATAR')
        Airline.airlinesName = input('Name the Airlines that you prefer to travel:
')
        self.flightNo = flightNo
#This method prints the details of the airlines
    def print_details(self):
        print('Airlines : ', Airline.airlinesName)
        print('Flight Number : ', self.flightNo)
        print(Airline.source, '-->', Airline.destination)
#Employee class Inheriting properties from the Airline class
class Employee(Airline):
    def __init__(self, employee_id, employee_name, employee_gender):
        self.employee_name = employee_name
        self.employee_id = employee_id
        self.employee_gender = employee_gender
#Method Overriding to print the Employee details
    def print_details(self):
        print("Name of employee: ", self.employee_name)
        print('Employee id: ', self.employee_id)
        print('Employee gender: ', self.employee_gender)
#This class inputs all the Traveller details
class Traveller:
    def __init__(self):
        Traveller.traveller_fn = input('Enter first name : ')
        Traveller.traveller_ln = input('Enter last name : ')
        Traveller.traveller_PNo = input('Enter passport number: ')
        Traveller.traveller_gender = input('Enter gender : ')
        Traveller.traveller_class = input('Business or Economy class? : ')
```

```
#This class is used to calculate the baggage fare
class Baggage:
    def __init__(self):
        Baggage.numberOfBags = int(input('Number of bags you want to checkin :
'))
        Baggage.totalBagFare =0;
        Baggage.numberOfBags = Baggage.numberOfBags
        if(Baggage.numberOfBags > 3):
            for i in range(Baggage.numberOfBags-3):
                Baggage.totalBagFare += 80
        print('You can take two bags for free !!! Total bag fare is for ',
Baggage.numberOfBags,'is ',Baggage.totalBagFare)
#This class calculates the ticket cost based on the class, flight and bags
class TicketCost(Baggage, Traveller, Airline):
    def __init__(self):
        TicketCost.baseCost = 250
        TicketCost.baseCost = TicketCost.baseCost + Baggage.totalBagFare
        if(Traveller.traveller_class == 'business'):
            TicketCost.baseCost = TicketCost.baseCost + 250
        print('Total ticket cost is : ',TicketCost.baseCost)
#These details are displayed on the ticket by using this method
    def ticketDisplay(self):
        print('Ticket Details')

print('**********************************************************************
***')
        print('Traveller Name : ',Traveller.traveller_fn,' ', Traveller.traveller_ln)
        print('Traveller Passport Number : ',Traveller.traveller_PNo)
        print('Gender : ', Traveller.traveller_gender)
        print('Class : ', Traveller.traveller_class)
        print('Total number of bags checked in : ', Baggage.numberOfBags)
        print('Total Fare for the trip is : ',TicketCost.baseCost)
employee = Employee(5555, 'haneesh', 'male')
employee.print_details()
flight = Airline('RX1006')
traveller = Traveller()
bags = Baggage()
ticket = TicketCost()
ticket.ticketDisplay()
flight.print_details()
```

4th Solution:

```
# Importing requests and beautifulsoup4 package

import requests
from bs4 import BeautifulSoup
url = 'https://catalog.umkc.edu/course-offerings/graduate/comp-sci/'
res = requests.get(url)
html_page = res.content

# converting the web page content to plain text
soup = BeautifulSoup(html_page, 'html.parser')
text = soup.find_all(text=True)
# Finding all the divs with class courseblock
res = soup.find_all('div', {'class': 'courseblock'})
# iterating through the res
for a in res:
  res1 = a.find('span', {'class': 'code'}).text
  res2 = a.find('p', {'class': 'courseblockdesc'}).text
# Printing the course code
  print(res1)
# printing the course description
  print(res2)
```

5th Solution:

```
from google.colab import drive
drive.mount('/content/drive/')
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
data=pd.read_csv('drive/My Drive/Lab1/cars.csv', delimiter=',', header=None, skip
rows=1, names=['mpg','cylinders','cubicinches','hp','weightlbs','time-to-
60','year','brand'])

# Nulls Handling
nulls = pd.DataFrame(data.isnull().sum().sort_values(ascending=False))
nulls.columns = ['Features']
nulls.index.name = 'Nulls_count'
print(nulls)
```

```
x = data.select_dtypes(include=[np.number]).interpolate().dropna()
print(sum(x.isnull().sum()  != 0))

# Encoding non-numeric features
from sklearn.preprocessing import LabelEncoder
data = data.apply(LabelEncoder().fit_transform)

# Here we are filling the null values with mean value
data=data.apply(lambda x: x.fillna(x.mean()),axis=0)
print(data["brand"])
print(data.isnull().sum())

# Visualize data to analyze our features correlations
import seaborn as sns
sns.set(style="white", color_codes=True)
import matplotlib.pyplot as plt
sns.FacetGrid(data, hue='brand', height=4).map(plt.scatter, 'mpg', 'cylinders').add_l
egend()
plt.show()
sns.FacetGrid(data, hue='brand', height=4).map(plt.scatter, 'cubicinches', 'hp').add_
legend()
plt.show()
sns.FacetGrid(data, hue='brand', height=4).map(plt.scatter, 'weightlbs', 'time-to-
60').add_legend()
plt.show()

# Split data into train and test
from sklearn.model_selection import train_test_split
x = data.drop(['brand'], axis=1)
y = data['brand']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =0.2, random_state=
0)

# KNN method
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
# Evaluate model
score = round(knn.score(x_train, y_train)* 100, 2)
```

```
print('K-Neighbors accuracy training score: ', score)
print('Classification report:')
y_pred = knn.predict(x_test)
print(classification_report(y_test, y_pred))


# Naive Bayes method
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
nb = GaussianNB()
nb.fit(x_train, y_train)
# Evaluate model
score = round(nb.score(x_train, y_train)* 100, 2)
print('Naive Bayes accuracy training score: ', score)
print('Classification report:')
y_pred = nb.predict(x_test)
print(classification_report(y_test, y_pred))


# SVM method
from sklearn.svm import SVC, LinearSVC
svc = SVC()
svc.fit(x_train, y_train)
# Evaluate model
score = round(svc.score(x_train, y_train)* 100, 2)
print('Support Vector Machines score: ', score)
print('Classification report:')
y_pred = svc.predict(x_test)
print(classification_report(y_test, y_pred))
```

6th_Solution:

```
from google.colab import drive
drive.mount('/content/drive')
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
```

```
data = pd.read_csv('/content/drive/My Drive/Lab1/cars.csv', delimiter=',', header=
None, skiprows=1, names=['mpg','cylinders','cubicinches','hp','weightlbs','time-to-
60','year','brand'])
# Nulls Handling
nulls = pd.DataFrame(data.isnull().sum().sort_values(ascending=False))
nulls.columns = ['Features']
nulls.index.name = 'Nulls_count'
print(nulls)
x = data.select_dtypes(include=[np.number]).interpolate().dropna()
print(sum(x.isnull().sum()  != 0))

# Encoding non-numeric features
from sklearn.preprocessing import LabelEncoder
data = data.apply(LabelEncoder().fit_transform)

# Here we are filling the null values with mean value
data=data.apply(lambda x: x.fillna(x.mean()),axis=0)
print(data["brand"])
print(data.isnull().sum())

# Visualize data to analyze our features correlations
import seaborn as sns
sns.set(style="white", color_codes=True)
import matplotlib.pyplot as plt
sns.FacetGrid(data, hue='brand', height=4).map(plt.scatter, 'mpg', 'cylinders').add_l
egend()
plt.show()
sns.FacetGrid(data, hue='brand', height=4).map(plt.scatter, 'cubicinches', 'hp').add_
legend()
plt.show()
sns.FacetGrid(data, hue='brand', height=4).map(plt.scatter, 'weightlbs', 'time-to-
60').add_legend()
plt.show()

#Apply k-means algorithm
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 9):
    km = KMeans(n_clusters=i, init='k-
means++', max_iter=300, n_init=10, random_state=0)
```

```
    km.fit(x)
    wcss.append(km.inertia_)
#Visualize elbow method
import matplotlib.pyplot as plt
plt.plot(range(1, 9), wcss)
plt.title = 'The Elbow Method'
plt.xlabel = 'n-clusters'
plt.ylabel = 'wcss'
plt.show()

#Found k=2
km = KMeans(n_clusters=2)
from sklearn.metrics import silhouette_score
km.fit(x)
x_pred = km.predict(x)
print('Silhouette score for k=2:', silhouette_score(x, x_pred))
#Found k=3
km = KMeans(n_clusters=3)
from sklearn.metrics import silhouette_score
km.fit(x)
x_pred = km.predict(x)
print('Silhouette score for k=3:', silhouette_score(x, x_pred))
#Found k=4
km = KMeans(n_clusters=4)
from sklearn.metrics import silhouette_score
km.fit(x)
x_pred = km.predict(x)
print('Silhouette score for k=4:', silhouette_score(x, x_pred))
```

7th_Solution:

```
from google.colab import drive
drive.mount('/content/drive/')
import nltk
nltk.download('punkt')
# (a) Read the data from a file.
file = open("/content/drive/My Drive/Lab1/nlp_input.txt","r",encoding='cp1252')
data=file.read()
print(data)
```

```
# (b) Tokenize the text into words and apply lemmatization technique on each wor
d.
# Tokenizing into words
wordtokens = nltk.word_tokenize(data)
for wt1 in wordtokens:
    print(wt1)
# Lemmatization
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
for wt2 in wordtokens:
    print(lemmatizer.lemmatize(wt2))
# (c) Find all the trigrams for the words.
from nltk.util import ngrams
trigramoutput = []
trigrams=ngrams(wordtokens,3)
for t in trigrams:
  trigramoutput.append(t)
print(trigramoutput)

# (d) Extract the top 10 of the most repeated trigrams based on their count.
wordfrequency = nltk.FreqDist(trigramoutput)
# Printing the most common words
commonwords = wordfrequency.most_common()
print("Trigrams Frequency : \n", commonwords)
# Top 10 Trigrams
top10 = wordfrequency.most_common(10)
print("Top 10 Trigrams : \n", top10)
# (e,f,g,h) Getting sentences using sentence tokenization.
sentencetokens = nltk.sent_tokenize(data)
# Creating an empty Array to append the sentences.
concatenated_result = []
for sentence in sentencetokens:
  for a,b,c in trigramoutput:
    for((d,e,f),length) in top10:
      if(a,b,c==d,e,f):
        concatenated_result.append(sentence)
print("concatenated result : ",concatenated_result)
```

8th Solution:

```
from google.colab import drive
drive.mount('/content/drive/')
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
data=pd.read_csv('drive/My Drive/Lab1/glass.csv')
x=data.drop(['Type'],axis=1)
y=data[['Type']]

# Training the Model with the data from glass dataset
from sklearn import linear_model
Mulreg = linear_model.LinearRegression()
Mulreg.fit(x, y)

# Before applying Exploratory data analysis(EDA)
from sklearn.metrics import mean_squared_error, r2_score
y_pred=Mulreg.predict(x)
print("R^2: %.2f" % r2_score(y,y_pred))
print("RMSE: %.2f" % mean_squared_error(y,y_pred))
train_data=pd.read_csv('drive/My Drive/Lab1/glass.csv')

# Nulls Handling
nulls = pd.DataFrame(train_data.isnull().sum().sort_values(ascending=False))
nulls.columns = ['Features']
nulls.index.name = 'Nulls_count'
print(nulls)
x = train_data.select_dtypes(include=[np.number]).interpolate().dropna()
print(sum(x.isnull().sum()  != 0))

# Here we are filling the null values with mean value
train_data=train_data.apply(lambda x: x.fillna(x.mean()),axis=0)
print(train_data["Type"])
print(train_data.isnull().sum())

# Split data into train and test
from sklearn.model_selection import train_test_split
```

```
x_train = train_data.drop(['Type'], axis=1)
y_train = train_data['Type']
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size =0.4, ran
dom_state=0)

# Training the Model with the data from glass dataset
from sklearn import linear_model
Mulreg = linear_model.LinearRegression()
Mulreg.fit(x_train, y_train)

# Before applying Exploratory data analysis(EDA)
from sklearn.metrics import mean_squared_error, r2_score
y_pred=Mulreg.predict(x_train)
print("R^2: %.2f" % r2_score(y_train,y_pred))
print("RMSE: %.2f" % mean_squared_error(y_train,y_pred))
```