Assignment 1

```
#import required library
import pandas as pd
import numpy as np
```

```
#read the dataset
dataset = pd.read_csv('Titanic.csv')
dataset.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.829
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.000

```
#check the dataset information
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Survived     418 non-null    int64  
 2   Pclass       418 non-null    int64  
 3   Name         418 non-null    object 
 4   Sex          418 non-null    object 
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64  
 7   Parch        418 non-null    int64  
 8   Ticket       418 non-null    object 
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object 
 11  Embarked     418 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

```
#describe the dataset
dataset.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	F
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.627
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.907
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.895
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.454
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.500
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329

We will check the null values in dataset

```
dataset.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
```

```
Embarked         0
dtype: int64
```

Imputing missing values of 'Age' column

```
#lets find the mean of the 'Age' column
d1 = dataset['Age'].mean()
d1
```

```
30.272590361445783
```

```
#find the round value of d1
d1 = round(d1)
d1
```

```
30
```

```
#fill the rounded value with missing value using fillna() method
dataset['Age'] = dataset['Age'].fillna(d1)
```

```
#check the result is affected or not
dataset.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
#do the same process for anothe columns that are missing value
d2 = round(dataset['Fare'].mean())
dataset['Fare'] = dataset['Fare'].fillna(d2)
dataset.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

```
#same for 'Cabin'
dataset['Cabin'].unique()
```

```
array([nan, 'B45', 'E31', 'B57 B59 B63 B66', 'B36', 'A21', 'C78', 'D34',
       'D19', 'A9', 'D15', 'C31', 'C23 C25 C27', 'F G63', 'B61', 'C53',
       'D43', 'C130', 'C132', 'C101', 'C55 C57', 'B71', 'C46', 'C116',
       'F', 'A29', 'G6', 'C6', 'C28', 'C51', 'E46', 'C54', 'C97', 'D22',
       'B10', 'F4', 'E45', 'E52', 'D30', 'B58 B60', 'E34', 'C62 C64',
       'A11', 'B11', 'C80', 'F33', 'C85', 'D37', 'C86', 'D21', 'C89',
       'F E46', 'A34', 'D', 'B26', 'C22 C26', 'B69', 'C32', 'B78',
       'F E57', 'F2', 'A18', 'C106', 'B51 B53 B55', 'D10 D12', 'E60',
       'E50', 'E39 E41', 'B52 B54 B56', 'C39', 'B24', 'D28', 'B41', 'C7',
       'D40', 'D38', 'C105'], dtype=object)
```

```
#lets count the value
dataset['Cabin'].value_counts()
```

```
B57 B59 B63 B66    3
B45                2
C89                2
C55 C57            2
A34                2
                  ..
E52                1
D30                1
```

```
E31                1
C62 C64            1
C105               1
Name: Cabin, Length: 76, dtype: int64
```

```
#‘ffill’ stands for ‘forward fill’ and will propagate last valid observation forward.
dataset['Cabin'] = dataset['Cabin'].ffill()
dataset.isnull().sum()
```

```
PassengerId     0
Survived        0
Pclass          0
Name            0
Sex             0
Age             0
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin          12
Embarked        0
dtype: int64
```

```
#bfill() will backward fill the NaN values that are present in the pandas dataframe.
dataset['Cabin'] = dataset['Cabin'].bfill()
dataset.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

```
#lets check our preprocessed dataset without any missing values
dataset.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
0	892	0	3	Kelly, Mr. James	male	34.5	0	0	330911	7.82
1	893	1	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.00

Assignment 2

```
#import the requred library
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
```

```
#now we'll get the dataset ready
from sklearn.datasets import fetch_20newsgroups
```

```
#define the categories which we want to classify
categories = ['rec.motorcycles', 'sci.electronics', 'comp.graphics', 'sci.med']

#sklearn provide us with subset for training and testig
train_data = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_state=42)
```

```
#lets check the target name
train_data.target_names
```

```
['comp.graphics', 'rec.motorcycles', 'sci.electronics', 'sci.med']
```

```
#test data
test_data = fetch_20newsgroups(subset='test', shuffle=True, random_state=42, categories= categories)
docs_test = test_data.data
```

```
#Builds a dictionary of features and transforms documents to feature vectors and convert our text documents to a matrix of token counts (
count_vect = CountVectorizer()

#train the model
x_train_count = count_vect.fit_transform(train_data.data)

#transform a count matrix to a normalized tf-idf representation
tfidf_transform = TfidfTransformer()
x_train_tfidf = tfidf_transform.fit_transform(x_train_count)
```

```
#fit our multinominal naive bayes classifier on train data to train it
knn = KNeighborsClassifier(n_neighbors=7)
```

```
#training our classifier
clf = knn.fit(x_train_tfidf, train_data.target)
```

```
#input data to predict their clsses of given categorie
doc_new = ['I have Harlay Davidson and Yamaha.', 'I have a GTX 1050 GPU']

#building up feature vector of our input
x_new_count = count_vect.transform(doc_new)

#we call transform instead of fit_transform beacause it's already been fit
x_new_tfidef = tfidf_transform.transform(x_new_count)
```

```
#predict the categories of our input text
predicted = clf.predict(x_new_tfidef)

for doc, categories in zip(doc_new, predicted):
  print('%r ==> %s'%(doc,train_data.target_names[categories]))
```

```
'I have Harlay Davidson and Yamaha.' ==> rec.motorcycles
'I have a GTX 1050 GPU' ==> sci.med
```

```
#finally evolute model by predicting the test data
text_clf = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf', knn),])
```

```
#predict test data
predicted = text_clf.predict(docs_test)
print(f'We got the accuracy of {np.mean(predicted == test_data.target)*100} % over the test data.')
```

```
We got the accuracy of 82.67766497461929 % over the test data.
```

Assignment 3

```
!pip install vaderSentiment
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 3.1 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from vaderSentiment) (2.27.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->vaderSentiment) (3.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests->vaderSentiment)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests->vaderSentiment) (2022.1
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests->vaderSentiment) (1.2
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
#import required libraries
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```
#create object of SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()
```

```
sentences = ["The book was good.",  # positive sentence
             "At least it isn't a horrible book.",  # negated negative sentence with contraction
             "The book was only kind of good.", # qualified positive sentence is handled correctly (intensity adjusted)
             "Today SUX!",  # negative slang with capitalization emphasis
             "Today only kinda sux! But I'll get by, lol", # mixed sentiment example with slang and constrastive conjunction "but"
             "Make sure you :) or :D today!",  # emoticons handled
             "Not bad at all"  # Capitalized negation
             ]
```

```
for sentence in sentences:
    vs = analyzer.polarity_scores(sentence)
    print("{:-<65} {}".format(sentence, str(vs)))
```

```
The book was good.------------------------------------------------ {'neg': 0.0, 'neu': 0.508, 'pos': 0.492, 'compound': 0.4404}
At least it isn't a horrible book.------------------------------- {'neg': 0.0, 'neu': 0.678, 'pos': 0.322, 'compound': 0.431}
The book was only kind of good.---------------------------------- {'neg': 0.0, 'neu': 0.697, 'pos': 0.303, 'compound': 0.3832}
Today SUX!------------------------------------------------------- {'neg': 0.779, 'neu': 0.221, 'pos': 0.0, 'compound': -0.5461}
Today only kinda sux! But I'll get by, lol----------------------- {'neg': 0.127, 'neu': 0.556, 'pos': 0.317, 'compound': 0.5249}
Make sure you :) or :D today!------------------------------------ {'neg': 0.0, 'neu': 0.294, 'pos': 0.706, 'compound': 0.8633}
Not bad at all--------------------------------------------------- {'neg': 0.0, 'neu': 0.513, 'pos': 0.487, 'compound': 0.431}
```

Assignment 5

```
#import packages
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
#create dataset using DataFrame
df=pd.DataFrame({
    'Points':['P1','P2','P3','P4','P5','P6','P7','P8',],
    'X':[0.1,0.15,0.08,0.16,0.2,0.25,0.24,0.3],
    'y':[0.6,0.71,0.9,0.85,0.3,0.5,0.1,0.2]
    })
df
```

	Points	X	y
0	P1	0.10	0.60
1	P2	0.15	0.71
2	P3	0.08	0.90
3	P4	0.16	0.85
4	P5	0.20	0.30
5	P6	0.25	0.50
6	P7	0.24	0.10
7	P8	0.30	0.20

```
#lets check the dataframes
f1 = df['X'].values
f2 = df['y'].values
X = np.array(list(zip(f1, f2)))
print(X)
```

```
[[0.1  0.6 ]
 [0.15 0.71]
 [0.08 0.9 ]
 [0.16 0.85]
 [0.2  0.3 ]
 [0.25 0.5 ]
 [0.24 0.1 ]
 [0.3  0.2 ]]
```

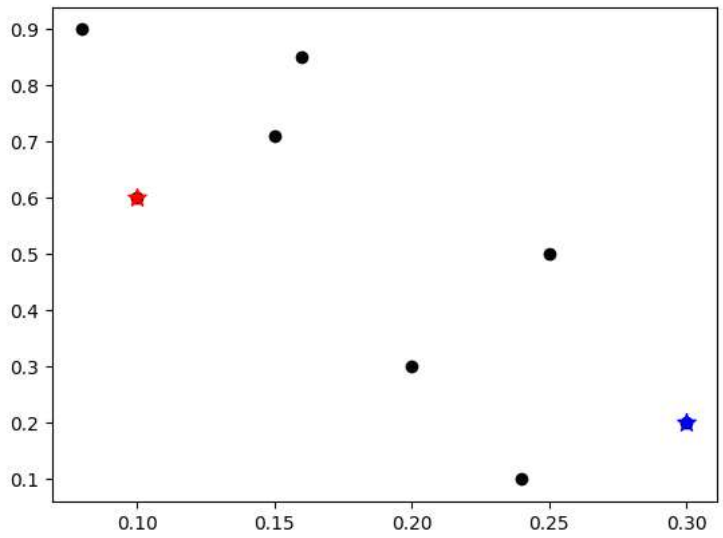
```
# initial centroid points
C_x=np.array([0.1,0.3])
C_y=np.array([0.6,0.2])
centroids=C_x,C_y
```

```
#plot the given points
colmap = {1: 'r', 2: 'b'}
plt.scatter(f1, f2, color='k')
plt.show()
```

```
#check the cluster
C = np.array(list((C_x, C_y)), dtype=np.float32)
print (C)
```

```
[[0.1 0.3]
 [0.6 0.2]]
```

```
#plot given elements with centroid elements
plt.scatter(f1, f2, c='#050505')
plt.scatter(C_x[0], C_y[0], marker='*', s=100, c='r')
plt.scatter(C_x[1], C_y[1], marker='*', s=100, c='b')
plt.show()
```




```
#import KMeans class and create object of it
from sklearn.cluster import KMeans
model=KMeans(n_clusters=2,random_state=0)
model.fit(X)
labels=model.labels_
print(labels)
```

```
[1 1 1 1 0 0 0 0]
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

```
#lets find out the population of cluster around m2
count=0
for i in range(len(labels)):
    if (labels[i]==1):
        count=count+1

print('No of population around cluster m2:',count-1)
```

```
No of population around cluster m2: 3
```

```
#Lets check the updated values of m1 and m2
new_centroids = model.cluster_centers_

print('Previous value of m1 and m2 is:')
print('M1==',centroids[0])
print('M1==',centroids[1])

print('updated value of m1 and m2 is:')
print('M1==',new_centroids[0])
print('M1==',new_centroids[1])
```

```
Previous value of m1 and m2 is:
M1== [0.1 0.3]
M1== [0.6 0.2]
updated value of m1 and m2 is:
M1== [0.2475 0.275 ]
M1== [0.1225 0.765 ]
```

✓ 0s completed at 3:37 PM