```
# Assignment number 1:
                # Load the dataset
                # Display basic information
                # Display statistical information
                # Display null values
                # Fill the null values
                # Change datatype of variable
                # Quantization (Encoding): Convert categorical to numerical variable
                # Normalization

#-------------------------------------------------------------------------------------

# Importing libraries
import pandas as pd
import numpy as np

#-------------------------------------------------------------------------------------

# Reading dataset
df = pd.read_csv('Placement.csv')

#-------------------------------------------------------------------------------------

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#-------------------------------------------------------------------------------------

# Display Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

# Fill the missing values
df['gender'].fillna(df['gender'].mode()[0], inplace=True)
df['ssc_p'].fillna(df['ssc_p'].mean(), inplace=True)
print('Mode of ssc_b: ', df['ssc_b'].mode())
df['ssc_b'].fillna(df['ssc_b'].mode()[0], inplace=True)

print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

# changing data type of columns
# see the datatype using df.dtypes
# change the datatype using astype

df['sl_no']=df['sl_no'].astype('int8')
print('Change in datatype: ', df['sl_no'].dtypes)
```

```
#-------------------------------------------------------------------------------------

# Converting categorical (qualitative) variable to numeric (quantitative) variable
                        # 1. Find and replace method
                        # 2. Label encoding method
                        # 3. OrdinalEncoder using scikit-learn

# Find and replace method
df['gender'].replace(['M','F'],[0,1],inplace=True)

# Label encoding method
df['ssc_b']=df['ssc_b'].astype('category')          #change data type to category
df['ssc_b']=df['ssc_b'].cat.codes

# Ordinal encoder using Scikit-learn
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
df[['hsc_b']]=enc.fit_transform(df[['hsc_b']])

print('After converting categorical variable to numeric variable: ')
print(df.head().T)
#-------------------------------------------------------------------------------------

# Normalization of data
# converting the range of data into uniform range
# marks [0-100]       [0-1]
# salary [200000 - 200000 per month] [0-1]

# Min-max feature scaling
# minimum value = 0
# maximum value = 1

# when we design model the higher value over powers in the model

df['salary']=(df['salary']-df['salary'].min())/(df['salary'].max()-df['salary'].min())
# (x - min value into that column)/(max value - min value)

# Maximum absolute scaler using scikit-learn
from sklearn.preprocessing import MaxAbsScaler
abs_scaler=MaxAbsScaler()
df[['mba_p']]=abs_scaler.fit_transform(df[['mba_p']])

#-------------------------------------------------------------------------------------

print(df.head().T)
```

```
# Assignment number 2:
                    # Load the dataset
                    # Display basic information
                    # Display statistical information
                    # Display null values
                    # Fill the null values
                    # Quantization (Encoding): Convert categorical to numerical variable
                    # Handle outliers
                    # Handle skewed data

#-------------------------------------------------------------------------------------

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#-------------------------------------------------------------------------------------

def DetectOutlier(df,var):

    # IQR method is used to deal with outliers


    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1

    high, low = Q3+1.5*IQR, Q1-1.5*IQR

    print("Highest allowed in variable:", var, high)
    print("lowest allowed in variable:", var, low)

    count = df[(df[var] > high) | (df[var] < low)][var].count()
    print('Total outliers in:',var,':',count)

    # new dataframe is created which contains outliers
    df1 = df[((df[var] < low) | (df[var] > high))]  #these are outliers
    print('Outliers : \n', len(df1))
    print(df1.T)

    df = df[((df[var] >= low) & (df[var] <= high))]  #now filter out data which is not outlier
    return(df)

#-------------------------------------------------------------------------------------

df = pd.read_csv('academic.csv')

#-------------------------------------------------------------------------------------

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------
```

```
# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#-------------------------------------------------------------------------------------

# Display Null values
print('Total Number of Null Values in Dataset: \n', df.isna().sum())

#-------------------------------------------------------------------------------------

# Fill the missing values
df['gender'].fillna(df['gender'].mode()[0], inplace=True)
df['raisedhands'].fillna(df['raisedhands'].mean(), inplace=True)
print('Total Number of Null Values in Dataset: \n', df.isna().sum())

#-------------------------------------------------------------------------------------

# Converting categorical to numeric using Find and replace method
df['Relation']=df['Relation'].astype('category')
df['Relation']=df['Relation'].cat.codes

#-------------------------------------------------------------------------------------

# Outliers can be visualized using boxplot
# using seaborn library we can plot the boxplot

fig, axes = plt.subplots(2,2)
fig.suptitle('Before removing Outliers')
sns.boxplot(data = df, x ='raisedhands', ax=axes[0,0])
sns.boxplot(data = df, x ='VisITedResources', ax=axes[0,1])
sns.boxplot(data = df, x ='AnnouncementsView', ax=axes[1,0])
sns.boxplot(data = df, x ='Discussion', ax=axes[1,1])
plt.show()

#Display and remove outliers
df = DetectOutlier(df, 'raisedhands')

fig, axes = plt.subplots(2,2)
fig.suptitle('After removing Outliers')
sns.boxplot(data = df, x ='raisedhands', ax=axes[0,0])
sns.boxplot(data = df, x ='VisITedResources', ax=axes[0,1])
sns.boxplot(data = df, x ='AnnouncementsView', ax=axes[1,0])
sns.boxplot(data = df, x ='Discussion', ax=axes[1,1])
plt.show()

#-------------------------------------------------------------------------------------

print('---------- Data Skew Values before Yeo John Transformation ----------')

# There are two types
# 1. Left skew
# 2. Right skew
# Formula to find out data skewness = 3*(mean-median)/std
#                                   = 0  (no skew) print
#                                   = negative (Negative skew)  left skewed data
#                                   = positve (Positive skew)   Right skewed data
#                                   = -0.5 to 0 to 0.5  (acceptable skew)
#                                   = -0.5> <-1 moderate negative skew
#                                   = 0.5> <1 moderate positive skew
#                                   = > -1 high negative
#                                   = > 1 high positive
```

```
print('raisedhands: ', df['raisedhands'].skew())
print('VisITedResources: ', df['VisITedResources'].skew())

print('AnnouncementsView: ', df['AnnouncementsView'].skew())
print('Discussion: ', df['Discussion'].skew())

fig, axes = plt.subplots(2,2)
fig.suptitle('Handling Data Skewness')
sns.histplot(ax = axes[0,0], data = df['AnnouncementsView'], kde=True)
sns.histplot(ax = axes[0,1], data = df['Discussion'], kde=True)


from sklearn.preprocessing import PowerTransformer
yeojohnTr = PowerTransformer(standardize=True)
df['AnnouncementsView'] =
yeojohnTr.fit_transform(df['AnnouncementsView'].values.reshape(-1,1))
df['Discussion'] = yeojohnTr.fit_transform(df['Discussion'].values.reshape(-1,1))


print('---------- Data Skew Values after Yeo John Transformation ----------')
print('AnnouncementsView: ', df['AnnouncementsView'].skew())
print('Discussion: ', df['Discussion'].skew())

sns.histplot(ax = axes[1,0], data = df['AnnouncementsView'], kde=True)
sns.histplot(ax = axes[1,1], data = df['Discussion'], kde=True)
plt.show()
```

```
# Assignment number 3:
                    # Load the dataset
                    # Display basic information
                    # Display null values
                    # Fill the null values
                    # Display overall statistical information
                    # Display groupwise statistical information

#-------------------------------------------------------------------------------------

import pandas as pd
import numpy as np

df = pd.read_csv('Employee_Salary.csv')

#-------------------------------------------------------------------------------------

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------

# Display Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

# Fill the missing values
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Experience'].fillna(df['Experience'].mean(), inplace=True)
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

# Display Overall Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#-------------------------------------------------------------------------------------
# groupwise statistical information
print('Groupwise Statistical Summary....')

print('\n------------------------- Experience ----------------------\n')
print(df['Experience'].groupby(df['Gender']).describe())

print('\n------------------------- Age ----------------------\n')
print(df['Age'].groupby(df['Gender']).describe())

print('\n------------------------- Salary ----------------------\n')
print(df['Salary'].groupby(df['Gender']).describe())


#-------------------------------------------------------------------------------------

df = pd.read_csv('iris.csv')
df = df.drop('Id', axis=1)
```

```
df.columns = ('SL', 'SW', 'PL', 'PW', 'Species')
print(df.head().T)

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())


print('Groupwise Statistical Summary....')

print('\n--------------------------- Sepal Length ----------------------\n')
print(df['SL'].groupby(df['Species']).describe())

print('\n--------------------------- Sepal Width ----------------------\n')
print(df['SW'].groupby(df['Species']).describe())

print('\n--------------------------- Petal Length ----------------------\n')
print(df['PL'].groupby(df['Species']).describe())

print('\n--------------------------- Petal Width ----------------------\n')
print(df['SW'].groupby(df['Species']).describe())
```

```
# Assignment number 4:
        # Load the Boston Housing dataset
        # Display basic information
        # Display statistical information
        # Display null values
        # Fill the null values
        # Feature Engineering through correlation matrix
        # Build the Linear Regression Model and find its accuracy score
        # Remove outliers and again see the accuracy of the model

#-------------------------------------------------------------------------------------

# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#-------------------------------------------------------------------------------------

def RemoveOutlier(df,var):

    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1
    high, low = Q3+1.5*IQR, Q1-1.5*IQR

    print("Highest allowed in variable:", var, high)
    print("lowest allowed in variable:", var, low)
    count = df[(df[var] > high) | (df[var] < low)][var].count()
    print('Total outliers in:',var,':',count)

    df = df[((df[var] >= low) & (df[var] <= high))]
    return df

#-------------------------------------------------------------------------------------

def BuildModel(X, Y):

    # 1. divide the dataset into training and testing  80%train 20%testing
    # 2. Choose the model (linear regression)
    # 3. Train the model using training data
    # 4. Test the model using testing data
    # 5. Improve the performance of the model

    # Training and testing data
    from sklearn.model_selection import train_test_split
    # Assign test data size 20%
    xtrain, xtest, ytrain, ytest =train_test_split(X,Y,test_size= 0.20, random_state=0)

   # Model selection and training
    from sklearn.linear_model import LinearRegression
    model = LinearRegression()
    model = model.fit(xtrain,ytrain)        #Training

    #Testing the model & show its accuracy / Performance
    ypred = model.predict(xtest)
    from sklearn.metrics import mean_absolute_error
    print('MAE:',mean_absolute_error(ytest,ypred))
    print("Model Score:",model.score(xtest,ytest))

#-------------------------------------------------------------------------------------
```

```
# Reading dataset
df = pd.read_csv('Boston.csv')

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe().T)

#-------------------------------------------------------------------------------------

# Display Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

# Feature Engineering - find out most relevant features to predict the output
# output is price of the house in boston housing dataset

# Display correlation matrix
sns.heatmap(df.corr(),annot=True)
plt.show()

# we observed that lstat, ptratio and rm have high correlation with cost of flat (medv)
# avoid variables which have more internal correlation
# lstat and rm have high internal correlation
# avoid lstat and rm together
#             1. lstat, ptratio
#             2. rm, ptratio
#             3. lstat, rm, ptratio
# #-------------------------------------------------------------------------------------

# Choosing input and output variables from correlation matrix
X = df[['ptratio','lstat']]     #input variables
Y = df['medv']                  #output variable

BuildModel(X, Y)


#-------------------------------------------------------------------------------------

# Checking model score after removing outliers
fig, axes = plt.subplots(1,2)
sns.boxplot(data = df, x ='ptratio', ax=axes[0])
sns.boxplot(data = df, x ='lstat', ax=axes[1])
fig.tight_layout()
plt.show()

df = RemoveOutlier(df, 'ptratio')
df = RemoveOutlier(df, 'lstat')

# Choosing input and output variables from correlation matrix
```

```
X = df[['ptratio','lstat']]
Y = df['medv']

BuildModel(X, Y)


# after feature engineering selecting 3 variables
# Choosing input and output variables from correlation matrix
X = df[['rm','lstat', 'ptratio']]
Y = df['medv']

BuildModel(X, Y)
```

```
# Assignment number 5:
        # Load the Social network ads dataset
        # Display basic information
        # Display statistical information
        # Display null values
        # Fill the null values
        # Feature Engineering through correlation matrix
        # Build the Logistic Regression Model and find its classification score
        # Remove outliers and again see the accuracy of the model

#-------------------------------------------------------------------------------------

# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#-------------------------------------------------------------------------------------

def RemoveOutlier(df,var):
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1
    high, low = Q3+1.5*IQR, Q1-1.5*IQR

    print("Highest allowed in variable:", var, high)
    print("lowest allowed in variable:", var, low)
    count = df[(df[var] > high) | (df[var] < low)][var].count()
    print('Total outliers in:',var,':',count)

    df = df[((df[var] >= low) & (df[var] <= high))]
    return df

#-------------------------------------------------------------------------------------

def BuildModel(X, Y):
    # Training and testing data
    from sklearn.model_selection import train_test_split

    # Assign test data size 20%
    xtrain, xtest, ytrain, ytest =train_test_split(X,Y,test_size= 0.25, random_state=13)

    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression(solver = 'lbfgs')
    model = model.fit(xtrain,ytrain)

    ypred = model.predict(xtest)

    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(ytest, ypred)
    sns.heatmap(cm, annot=True)
    plt.show()

    from sklearn.metrics import classification_report
    print(classification_report(ytest, ypred))

#-------------------------------------------------------------------------------------

# Reading dataset
df = pd.read_csv('purchase.csv')
```

```
# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

df = df.drop('User ID', axis=1)
df.columns = ['Gender', 'Age', 'Salary', 'Purchased']
#--------------------------------------------------------------------------------------

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#--------------------------------------------------------------------------------------

# Display Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())

#--------------------------------------------------------------------------------------

# Label encoding method
df['Gender']=df['Gender'].astype('category')
df['Gender']=df['Gender'].cat.codes


# Display correlation matrix
sns.heatmap(df.corr(),annot=True)
plt.show()

#--------------------------------------------------------------------------------------

# Choosing input and output variables from correlation matrix
X = df[['Age','Salary']]
Y = df['Purchased']

BuildModel(X, Y)

# #--------------------------------------------------------------------------------------

# Checking model score after removing outliers
fig, axes = plt.subplots(1,2)
sns.boxplot(data = df, x ='Age', ax=axes[0])
sns.boxplot(data = df, x ='Salary', ax=axes[1])
fig.tight_layout()
plt.show()

df = RemoveOutlier(df, 'Age')
df = RemoveOutlier(df, 'Salary')

# You can use normalization method to improve the score
# salary -> high range
# age -> low range
# Normalization will smoothe both range salary and age

# Choosing input and output variables from correlation matrix
X = df[['Age','Salary']]
Y = df['Purchased']
BuildModel(X, Y)
```

```
# Assignment number 6:
        # Load the Iris dataset
        # Display basic information
        # Display statistical information
        # Display null values
        # Fill the null values
        # Feature Engineering through correlation matrix
        # Build the Gaussian Naive Bayes Model and find its classification score
        # Remove outliers and again see the accuracy of the model

#-------------------------------------------------------------------------------------

# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#-------------------------------------------------------------------------------------

def RemoveOutlier(df,var):
    Q1 = df[var].quantile(0.25)
    Q3 = df[var].quantile(0.75)
    IQR = Q3 - Q1
    high, low = Q3+1.5*IQR, Q1-1.5*IQR

    print("Highest allowed in variable:", var, high)
    print("lowest allowed in variable:", var, low)
    count = df[(df[var] > high) | (df[var] < low)][var].count()
    print('Total outliers in:',var,':',count)

    df = df[((df[var] >= low) & (df[var] <= high))]
    return df

#-------------------------------------------------------------------------------------

def BuildModel(X, Y):
    # Training and testing data
    from sklearn.model_selection import train_test_split

    # Assign test data size 20%
    xtrain, xtest, ytrain, ytest =train_test_split(X,Y,test_size= 0.25, random_state=0)

    # from sklearn.linear_model import LogisticRegression
    # model = LogisticRegression(solver = 'lbfgs')

    from sklearn.naive_bayes import GaussianNB
    model = GaussianNB()
    model = model.fit(xtrain,ytrain)

    ypred = model.predict(xtest)

    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(ytest, ypred)
    sns.heatmap(cm, annot=True)
    plt.show()

    from sklearn.metrics import classification_report
    print(classification_report(ytest, ypred))

#-------------------------------------------------------------------------------------
```

```
# Reading dataset
df = pd.read_csv('iris.csv')
df = df.drop('Id', axis=1)
df.columns = ('SL', 'SW', 'PL', 'PW', 'Species')

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#-------------------------------------------------------------------------------------

# Display Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

# Label encoding method
df['Species']=df['Species'].astype('category')
df['Species']=df['Species'].cat.codes


# Display correlation matrix
sns.heatmap(df.corr(),annot=True)
plt.show()

#-------------------------------------------------------------------------------------

# Choosing input and output variables from correlation matrix
X = df[['SL','SW', 'PL', 'PW']]
Y = df['Species']
BuildModel(X, Y)

 #-------------------------------------------------------------------------------------

# Checking model score after removing outliers
fig, axes = plt.subplots(2,2)
sns.boxplot(data = df, x ='SL', ax=axes[0,0])
sns.boxplot(data = df, x ='SW', ax=axes[0,1])
sns.boxplot(data = df, x ='PL', ax=axes[1,0])
sns.boxplot(data = df, x ='PW', ax=axes[1,1])
plt.show()

df = RemoveOutlier(df, 'SW')

# Choosing input and output variables from correlation matrix
X = df[['SL','SW', 'PL', 'PW']]
Y = df['Species']

BuildModel(X, Y)
#After removing outliers accuracy is reducing due to overfitting of the model
```

```
# Assignment No. 7: Text Analytics
                # 1. Tokenization (Sentence, Word),
                # 2. POS Tagging,
                # 3. Stop words removal,
                # 4. Stemming and Lemmatization.
                # 5. Calculate TF, IDF, TF-IDF
#-------------------------------------------------------------------------------------------------------------------------

import pandas as pd
import nltk                             #natural language tool kit library widely used for NLP
applications
import re                               # regular expression used for pattern matching

# nltk.download('punkt')
# nltk.download('stopwords')
# nltk.download('wordnet')
# nltk.download('averaged_perceptron_tagger')

#-------------------------------------------------------------------------------------------------------------------------

def computeTF(wordDict, bagOfWords):
        tfDict = {}
        bagOfWordsCount = len(bagOfWords)
        for word, count in wordDict.items():
                tfDict[word] = count / float(bagOfWordsCount)
        return tfDict

#-------------------------------------------------------------------------------------------------------------------------

def computeIDF(documents):
        import math
        N = len(documents)
        idfDict = dict.fromkeys(documents[0].keys(), 0)
        for document in documents:
                for word, val in document.items():
                        if val > 0:
                                idfDict[word] += 1
                                for word, val in idfDict.items():
                                        if val > 0: idfDict[word] = math.log(N / float(val))
                                        else: idfDict[word] = 0
        return idfDict

#-------------------------------------------------------------------------------------------------------------------------

def computeTFIDF(tfBagOfWords, idfs):
        tfidf = {}
        for word, val in tfBagOfWords.items():
                tfidf[word] = val * idfs[word]
        return tfidf

text= "Tokenization is the first step in text analytics. The process of breaking down a text
paragraph into smaller chunks such as words or sentences is called Tokenization."
print('The given sentences are: \n', text)

#-------------------------------------------------------------------------------------------------------------------------

#Sentence Tokenization
from nltk.tokenize import sent_tokenize
tokenized_text= sent_tokenize(text)
print("\n Sentence Tokenization: \n", tokenized_text)

#Word Tokenization
```

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print('\nWord Tokeniztion: \n', tokenized_word)

#-----------------------------------------------------------------------------------------------------------------------

# Add code for POS Tagging

#-----------------------------------------------------------------------------------------------------------------------

from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))

# Removing stop words
text= "How to remove stop words with NLTK library in Python?"
text= re.sub('[^a-zA-Z]', ' ',text)
tokens = word_tokenize(text.lower())
filtered_text=[]
for w in tokens:
        if w not in stop_words:
                filtered_text.append(w)

print ("Tokenized Sentence:",tokens)
print ("Filterd Sentence:",filtered_text)

#-----------------------------------------------------------------------------------------------------------------------

#Stamming
from nltk.stem import PorterStemmer
e_words= ["wait", "waiting", "waited", "waits"]
ps =PorterStemmer()
for w in e_words:
        rootWord=ps.stem(w)
        print('Stemming for ',w,': ',rootWord)

#Lemmatization
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
        print("Lemma for {} is {}".format(w,
        wordnet_lemmatizer.lemmatize(w)))

#-----------------------------------------------------------------------------------------------------------------------

# Algorithm for Create representation of document by calculating TFIDF

# Step 1: Import the necessary libraries.
from sklearn.feature_extraction.text import TfidfVectorizer

# Step 2: Initialize the Documents.
documentA = 'Jupiter is the largest planet'
documentB = 'Mars is the fourth planet from the Sun'

# Step 3: Create BagofWords (BoW) for Document A and B. word tokenization
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')

# Step 4: Create Collection of Unique words from Document A and B.
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

```
# Step 5: Create a dictionary of words and their occurrence for each document in the corpus
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
        numOfWordsA[word] += 1                    #How many times each word is repeated

numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
        numOfWordsB[word] += 1

# Step 6: Compute the term frequency for each of our documents.
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)

# Step 7: Compute the term Inverse Document Frequency.
print('----------------Term Frequency----------------------')
df = pd.DataFrame([tfA, tfB])
print(df)

# Step 8: Compute the term TF/IDF for all words.
idfs = computeIDF([numOfWordsA, numOfWordsB])
print('----------------Inverse Document Frequency----------------------')
print(idfs)

tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
print('------------------- TF-IDF-----------------------------------')
df = pd.DataFrame([tfidfA, tfidfB])
print(df)
```

```
# Assignment number 8:
                    # Load the Titanic dataset
                    # Display basic information
                    # Display statistical information
                    # Display null values
                    # Fill the null values
                    # Display and iterpret Histogram of one variable and two variables
#-------------------------------------------------------------------------------------

# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#-------------------------------------------------------------------------------------

# Reading dataset
df = pd.read_csv('titanic.csv')

#-------------------------------------------------------------------------------------

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#-------------------------------------------------------------------------------------

# Display and fill the Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())
df['Age'].fillna(df['Age'].median(), inplace=True)
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

# Single variable histogram
fig, axis = plt.subplots(1,3)
sns.histplot(ax = axis[0], data = df, x='Sex', hue = 'Sex', multiple = 'dodge', shrink = 0.8)
sns.histplot(ax = axis[1], data = df, x='Pclass', hue = 'Pclass',multiple = 'dodge', shrink = 0.8)
sns.histplot(ax = axis[2], data = df, x='Survived', hue = 'Survived', multiple = 'dodge', shrink = 0.8)
plt.show()


# Single variable histogram
fig, axis = plt.subplots(1,2)
sns.histplot(ax = axis[0], data = df, x='Age', multiple = 'dodge', shrink = 0.8, kde = True)
sns.histplot(ax = axis[1], data = df, x='Fare', multiple = 'dodge', shrink = 0.8, kde = True)
plt.show()


# Two variable histogram
```

```
fig, axis = plt.subplots(2,2)
sns.histplot(ax = axis[0,0], data = df, x='Age', hue = 'Sex', multiple = 'dodge', shrink = 0.8, kde = True)
sns.histplot(ax = axis[0,1], data = df, x='Fare', hue = 'Sex', multiple = 'dodge', shrink = 0.8, kde = True)
sns.histplot(ax=axis[1,0], data=df, x='Age', hue = 'Survived', multiple = 'dodge',shrink=0.8, kde= True)
sns.histplot(ax = axis[1,1], data=df, x='Fare', hue='Survived', multiple='dodge', shrink=0.8, kde = True)
plt.show()


# Two variable histogram
fig, axis = plt.subplots(2,2)
sns.histplot(ax=axis[0,0], data=df, x='Sex', hue='Survived', multiple= 'dodge', shrink = 0.8, kde = True)
sns.histplot(ax=axis[0,1], data=df, x='Pclass', hue='Survived', multiple='dodge', shrink=0.8, kde= True)
sns.histplot(ax=axis[1,0], data=df, x='Age', hue='Survived', multiple='dodge', shrink = 0.8, kde = True)
sns.histplot(ax=axis[1,1], data=df, x='Fare', hue='Survived', multiple='dodge', shrink = 0.8, kde = True)
plt.show()
```

```
# Assignment number 9:
                    # Load the Titanic dataset
                    # Display basic information
                    # Display statistical information
                    # Display null values
                    # Fill the null values
                    # Display and iterpret boxplot of one variable, two variables and three variables
#-------------------------------------------------------------------------------------

# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#-------------------------------------------------------------------------------------

# Reading dataset
df = pd.read_csv('titanic.csv')

#-------------------------------------------------------------------------------------

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#-------------------------------------------------------------------------------------

# Display and fill the Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())
df['Age'].fillna(df['Age'].median(), inplace=True)
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

#One variable
fig, axes = plt.subplots(1,2)
sns.boxplot(data = df, y ='Age', ax=axes[0])
sns.boxplot(data = df, y ='Fare', ax=axes[1])
plt.show()

# Two variables
fig, axes = plt.subplots(1,3, sharey=True)
sns.boxplot(data = df, x='Sex', y ='Age', hue = 'Sex', ax=axes[0])
sns.boxplot(data = df, x='Pclass', y ='Age', hue = 'Pclass', ax=axes[1])
sns.boxplot(data = df, x='Survived', y ='Age', hue = 'Survived', ax=axes[2])
plt.show()


# Two variables
fig, axes = plt.subplots(1,3, sharey=True)
```

```
sns.boxplot(data = df, x='Sex', y ='Fare', hue = 'Sex', ax=axes[0], log_scale = True)
sns.boxplot(data = df, x='Pclass', y ='Fare', hue = 'Pclass', ax=axes[1], log_scale = True)
sns.boxplot(data = df, x='Survived', y ='Fare', hue = 'Survived', ax=axes[2], log_scale = True)
plt.show()


#three variables
fig, axes = plt.subplots(1,2, sharey=True)
sns.boxplot(data = df, x='Sex', y ='Age', hue = 'Survived', ax=axes[0])
sns.boxplot(data = df, x='Pclass', y ='Age', hue = 'Survived', ax=axes[1])
plt.show()


fig, axes = plt.subplots(1,2, sharey=True)
sns.boxplot(data = df, x='Sex', y ='Fare', hue = 'Survived', ax=axes[0], log_scale = True)
sns.boxplot(data = df, x='Pclass', y ='Fare', hue = 'Survived', ax=axes[1], log_scale = True)
plt.show()
```

```
# Assignment number 10:
                    # Load the Iris dataset
                    # Display basic information
                    # Display statistical information
                    # Display null values
                    # Fill the null values
                    # Display and iterpret Boxplot & Histogram
                    # Identify outliers
#-------------------------------------------------------------------------------------

# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#-------------------------------------------------------------------------------------

# Reading dataset
df = pd.read_csv('iris.csv')
df = df.drop('Id', axis=1)
df.columns = ('SL', 'SW', 'PL', 'PW', 'Species')

#-------------------------------------------------------------------------------------

# Display basic information
print('Information of Dataset:\n', df.info)
print('Shape of Dataset (row x column): ', df.shape)
print('Columns Name: ', df.columns)
print('Total elements in dataset:', df.size)
print('Datatype of attributes (columns):', df.dtypes)
print('First 5 rows:\n', df.head().T)
print('Last 5 rows:\n',df.tail().T)
print('Any 5 rows:\n',df.sample(5).T)

#-------------------------------------------------------------------------------------

# Display Statistical information
print('Statistical information of Numerical Columns: \n',df.describe())

#-------------------------------------------------------------------------------------

# Display and fill the Null values
print('Total Number of Null Values in Dataset:', df.isna().sum())

#-------------------------------------------------------------------------------------

fig, axis = plt.subplots(2,2)
sns.boxplot(ax = axis[0,0], data = df, y='SL')
sns.boxplot(ax = axis[0,1], data = df, y='SW')
sns.boxplot(ax = axis[1,0], data = df, y='PL')
sns.boxplot(ax = axis[1,1], data = df, y='PW')
plt.show()

fig, axis = plt.subplots(2,2)
sns.boxplot(ax = axis[0,0], data = df, y='SL', hue='Species')
sns.boxplot(ax = axis[0,1], data = df, y='SW', hue='Species')
sns.boxplot(ax = axis[1,0], data = df, y='PL', hue='Species')
sns.boxplot(ax = axis[1,1], data = df, y='PW', hue='Species')
plt.show()

fig, axis = plt.subplots(2,2)
```

```
sns.histplot(ax = axis[0,0], data = df, x='SL', multiple = 'dodge', shrink = 0.8, kde = True)
sns.histplot(ax = axis[0,1], data = df, x='SW', multiple = 'dodge', shrink = 0.8, kde = True)
sns.histplot(ax = axis[1,0], data = df, x='PL', multiple = 'dodge', shrink = 0.8, kde = True)
sns.histplot(ax = axis[1,1], data = df, x='PW', multiple = 'dodge', shrink = 0.8, kde = True)
plt.show()
```

```
fig, axis = plt.subplots(2,2)
sns.histplot(ax=axis[0,0], data=df, x='SL', hue='Species', element='poly', shrink=0.8, kde= True)
sns.histplot(ax=axis[0,1], data = df, x='SW', hue = 'Species', element = 'poly', shrink = 0.8, kde = True)
sns.histplot(ax=axis[1,0], data = df, x='PL', hue = 'Species', element = 'poly', shrink = 0.8, kde = True)
sns.histplot(ax=axis[1,1], data = df, x='PW', hue = 'Species', element = 'poly', shrink = 0.8, kde = True)
plt.show()
```