FREQUENT ITEMSET AND ASSOCIATION RULE MINING

```
pip install mlxtend
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
transactions = [['bread', 'milk'],
                ['bread', 'diaper','beer','eggs'],
                ['milk','diaper','beer','coke'],
                ['bread', 'milk','diaper','beer'],
                ['bread','milk','diaper','coke']]
from mlxtend.preprocessing import TransactionEncoder
te=TransactionEncoder()
te_array=te.fit(transactions).transform(transactions)
df=pd.DataFrame(te_array, columns=te.columns_)
df
#Apriori Algorithm
freq_items = apriori(df, min_support = 0.5, use_colnames = True)
print(freq_items)
#Mining Association Rules
rules = association_rules(freq_items, metric ='confidence',
min_threshold=0.05)
rules = rules.sort_values(['support', 'confidence'], ascending
=[False,False])
rules
```

#Interpretation
#Change the min_support value and perform again.

If data is to be taken from csv file

```
import pandas as pd
data=pd.read_csv('Market_Basket_Optimisation.csv')
# Getting the list of transactions from the dataset
transactions = []
for i in range(0, len(data)):
 transactions.append([str(data.values[i,j]) for j in range(0,
len(data.columns))])

from mlxtend.preprocessing import TransactionEncoder
```

```
te=TransactionEncoder()
te_array=te.fit(transactions).transform(transactions)
df=pd.DataFrame(te_array, columns=te.columns_)
df

# Training Apriori algorithm on the dataset
!pip install apyori
from apyori import apriori

model=apriori(transactions, min_support = 0.003, min_confidence = 0.2,
min_lift = 3, min_length = 2,max_length = 3)
model_table=list(model)
pd.DataFrame(model_table)
```

LINEAR REGRESSION

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split from
sklearn.linear_model import LinearRegression from
sklearn.metrics import r2_score

data=pd.read_csv('sales_A1.csv')
data.info()
data.describe()
x= np.array(data[['TV']])
y= np.array(data[['Sales']])

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size =
0.7, test_size = 0.3, random_state = 100)
x_train.shape
x_test.shape
model = LinearRegression()
model.fit(x_train, y_train)
model.coef_
model.intercept_
y_pred=model.predict(x_test)
```

```
#Finding the value of coefficient of Determination
r2_score(y_test,y_pred)
```

Write regression equation and r square

LOGISTIC REGRESSION

data=pd.read_csv('C:\\User_Dataset.csv')

data.describe()

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size = 0.25, random_state = 25)
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
#Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Write accuracy and confusion matrix

```
!pip install nltk
        import nltk
```

In [ ]:
```
nltk.download('all')
```

In [ ]:
```
nltk.download('punkt')
nltk.download('stopwords')
```

In [12]:
```
text = """ Artificial intelligence is the future. Artificial
 intelligence is science fiction. Artificial intell
```

In [30]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
```

In [31]:
```
import re
import heapq
def clean_data(data):
 text = re.sub(r"\[[0-9]*\]"," ",data)
 text = text.lower()
 text = re.sub(r'\s+'," ",text)
 text = re.sub(r","," ",text)
 return text
cleaned_data = clean_data(text)

sent_tokens = sent_tokenize(cleaned_data)
word_tokens = word_tokenize(cleaned_data)
word_frequency = {}
stopwords = set(stopwords.words("english"))

for word in word_tokens:
 if word not in stopwords:
 if word not in word_frequency.keys():
 word_frequency[word]=1
 else:
 word_frequency[word] +=1
maximum_frequency = max(word_frequency.values())
for word in word_frequency.keys():
 word_frequency[word] = (word_frequency[word]/maximum_frequency)


sentences_score = {}
```

```
for sentence in sent_tokens:
 for word in word_tokenize(sentence):
 if word in word_frequency.keys():
 if (len(sentence.split(" "))) <30:
 if sentence not in sentences_score.keys():
 sentences_score[sentence] = word_frequency[word]
 else:
 sentences_score[sentence] += word_frequency[word]


def get_key(val):
 for key, value in sentences_score.items():
 if val == value:
 return key
key = get_key(max(sentences_score.values()))
summary = heapq.nlargest(5,sentences_score,key=sentences_score.get)
print(" ".join(summary))
```

```
artificial intelligence is already part of our everyday lives. and all
three are part of the reason why alphago trounced lee se-dol. all
those statements are true it just depends on what flavor of ai you are
referring to. artificial intelligence is science fiction. artificial
intelligence is the future.
```

Set A

Q2.

In [51]:
```
import nltk
 from nltk.corpus import stopwords
 from nltk.tokenize import sent_tokenize, word_tokenize
 from nltk.probability import FreqDist
```

In [52]:
```
text="""As the process of analyzing raw data to find trends and answer
           questions, the definition of data analyt
```

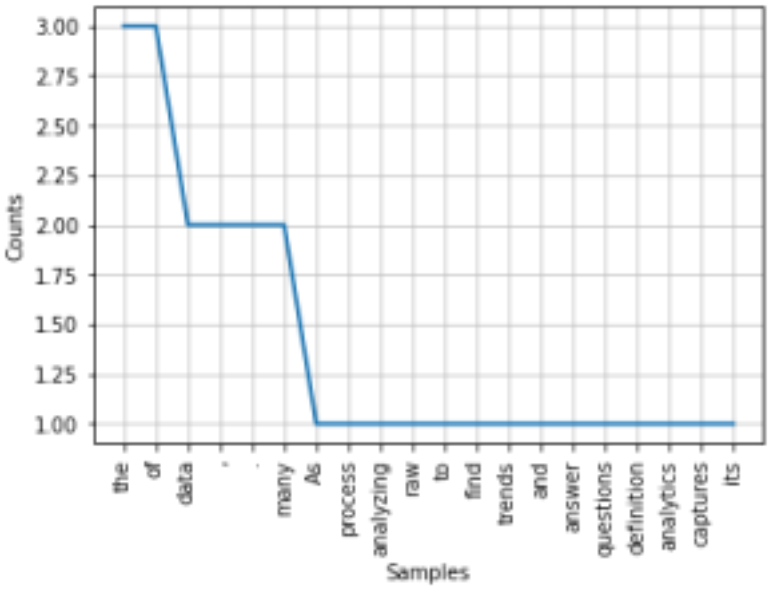
In [53]:
```
sent_tokens = sent_tokenize(text)
 word_tokens = word_tokenize(text)
 stopwords = set(stopwords.words("english"))

 frequency_distribution=FreqDist(word_tokens)
 print(frequency_distribution)
 frequency_distribution
```

```
<FreqDist with 30 samples and 38 outcomes>
```

Out[53]:
```
FreqDist({'the': 3, 'of': 3, 'data': 2, ',': 2, '.': 2,
          'many': 2, 'As': 1, 'process': 1, 'analyzi
          1, ...})
```

In [54]:
```
import matplotlib.pyplot as plt
 ax = plt.axes()
```

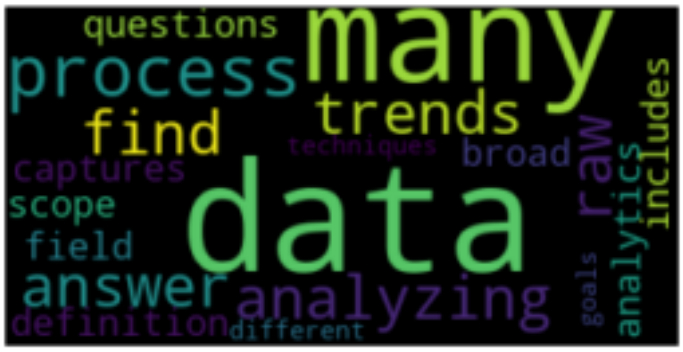
```
frequency_distribution.plot(20,cumulative=False) #first 20 words starting from
maximum onn Xaxis  ax.set_title('Frequency Plot of words')
plt.show()
```



In [ ]:
```
!pip install wordcloud
  from wordcloud import WordCloud
```

In [74]:
```
word_cloud=WordCloud(background_color='black').generate(text)
```

In [75]:
```
plt.figure()
  plt.imshow(word_cloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()
```



Set A

Q3.

In [ ]:
```
nltk.download('vader_lexicon')
```

In [77]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
  vader_analyzer=SentimentIntensityAnalyzer()
```

In [78]:
```
text1="I purchased headphones online. I am very happy with the product."
  print(vader_analyzer.polarity_scores(text1))
```

```
{'neg': 0.0, 'neu': 0.667, 'pos': 0.333, 'compound': 0.6115}
```

In [79]:
```
text2="I saw the movie yesterday. The animation was really good but
  the script was ok." print(vader_analyzer.polarity_scores(text2))
```

```
{'neg': 0.0, 'neu': 0.71, 'pos': 0.29, 'compound': 0.5989}
```

```
text="""As the process of analyzing raw data to find trends and answer questions, the definition of
data analyt
```

In [92]:
```
from nltk.tokenize import sent_tokenize, word_tokenize
  sent_tokens = sent_tokenize(text)
  word_tokens = word_tokenize(text)

  print("Tokenized Sentences : \n", sent_tokens, "\n") #break paragraph into sentences (.
  full stop is considere print("Tokenized Words : \n",word_tokens, "\n") #break sentences
  into words ( <spaces are considered > )
```

```
Tokenized Sentences :
 ['As the process of analyzing raw data to find trends and answer questions, the
definition of data analytics c aptures its broad scope of the field.', 'However, it
includes many techniques with many different goals.']

Tokenized Words :
 ['As', 'the', 'process', 'of', 'analyzing', 'raw', 'data', 'to', 'find', 'trends', 'and',
'answer', 'question s', ',', 'the', 'definition', 'of', 'data', 'analytics', 'captures',
'its', 'broad', 'scope', 'of', 'the', 'fie ld', '.', 'However', ',', 'it', 'includes',
'many', 'techniques', 'with', 'many', 'different', 'goals', '.']
```

In [95]:
```
from nltk.corpus import stopwords
  stopwords = set(stopwords.words("english"))
  print(stopwords)
```

```
{'or', 'about', 'above', 'not', 'o', 'than', 'wasn', 'yourselves', 'down', 'most',
"aren't", 'mustn', 'too', 's ame', 'such', 'weren', 'these', 'from', 'very', 'why',
"shan't", 'on', 'll', "hadn't", 'over', 'd', 'she', 'bel ow', 'them', 'through', 've',
'herself', 'don', 'for', 'be', 'until', 'a', "haven't", "wouldn't", 'they', 'thei rs',
'i', 'when', 'hadn', 'ma', 'few', 'by', 'hers', 'couldn', 'it', 'under', 'no', "she's",
```

```
'because', 'thei
r', 'should', 'just', 'after', 'he', 'what', 'will', 'needn', 'do', "it's", "won't",
'have', 'didn', 'did', 't', 'won', 'itself', "doesn't", 'while', 'himself', 're', 'nor',
'is', 'in', "isn't", 'once', 'so', 'ours', 'y ourself', 'hasn', 'only', 'yours', 'ain',
"wasn't", 'shouldn', "you'll", 'y', 'out', 'm', 'to', "couldn't", "yo u'd", 'are',
'doesn', 'between', 'any', 'aren', 'been', 'haven', 'my', 'we', 'here', 'who', "don't",
"that'll", 'which', 'more', 'into', 'if', "weren't", 'and', 'up', "shouldn't", 'again',
'of', 'against', 'him', 'other', 'ourselves', 'me', "you're", 'those', 'were', 'during',
"you've", 'but', 'an', 'where', 'was', 'then', 'does', "mightn't", 's', 'this',
"should've", 'your', 'before', 'both', 'had', 'the', 'further', 'having', 'can', 'is n',
'each', 'wouldn', 'with', "needn't", 'his', 'has', 'how', "didn't", 'being', 'as',
"hasn't", 'mightn', 'ou r', "mustn't", 'am', 'themselves', 'some', 'doing', 'at', 'own',
'you', 'whom', 'there', 'all', 'that', 'its', 'myself', 'shan', 'off', 'her', 'now'}
```

In [97]:
```
filtered_words_list=[]
 for words in word_tokens:
  if words not in stopwords:
  filtered_words_list.append(words)
 print("Tokenized Words : \n",word_tokens,"\n")
 print("Filtered Words : \n",filtered_words_list,"\n")
```

```
Tokenized Words :
 ['As', 'the', 'process', 'of', 'analyzing', 'raw', 'data', 'to', 'find', 'trends', 'and',
'answer', 'question s', ',', 'the', 'definition', 'of', 'data', 'analytics', 'captures',
'its', 'broad', 'scope', 'of', 'the', 'fie ld', '.', 'However', ',', 'it', 'includes',
'many', 'techniques', 'with', 'many', 'different', 'goals', '.']

Filtered Words :
 ['As', 'process', 'analyzing', 'raw', 'data', 'find', 'trends', 'answer', 'questions',
',', 'definition', 'dat a', 'analytics', 'captures', 'broad', 'scope', 'field', '.',
'However', ',', 'includes', 'many', 'techniques', 'many', 'different', 'goals', '.']
```

In [98]:
```
#Stemming change writing, wrote, written can stemmed or reduced as write
 from nltk.stem import PorterStemmer
 porter_stemmer=PorterStemmer()
 stemmed_text_words=[]
 for words in filtered_words_list:
  stemmed_text_words.append(porter_stemmer.stem(words))
 print("Filtered Words : \n",word_tokens,"\n")
 print("Stemmed Words : \n",stemmed_text_words,"\n")
```

```
Filtered Words :
 ['As', 'the', 'process', 'of', 'analyzing', 'raw', 'data', 'to', 'find', 'trends', 'and',
'answer', 'question s', ',', 'the', 'definition', 'of', 'data', 'analytics', 'captures',
'its', 'broad', 'scope', 'of', 'the', 'fie ld', '.', 'However', ',', 'it', 'includes',
'many', 'techniques', 'with', 'many', 'different', 'goals', '.']

Stemmed Words :
 ['as', 'process', 'analyz', 'raw', 'data', 'find', 'trend', 'answer', 'question', ',',
'definit', 'data', 'ana lyt', 'captur', 'broad', 'scope', 'field', '.', 'howev', ',',
'includ', 'mani', 'techniqu', 'mani', 'differ', 'goal', '.']
```

In [103...
```
#Stemming
from nltk.stem import PorterStemmer
PorterStemmer().stem('eaten')
```

Out[103...
```
'eaten'
```

In [104...
```
PorterStemmer().stem('eating')
```

Out[104...
```
'eat'
```

In [107...
```
PorterStemmer().stem('eated')
```

Out[107...
```
'eat'
```

Q. Find the lemmatized word for eaten.

In [ ]:
```
#Lemmatization
import nltk
nltk.download('wordnet')
```

In [106...
```
# Lemmatization
from nltk.stem.wordnet import WordNetLemmatizer
lemmatizer=WordNetLemmatizer()
word_text="eaten"
print("Lemmatized Word :

",lemmatizer.lemmatize(word_text,"v"))
```

```
Lemmatized Word : eat
```

.....