Installing Libraries

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import sklearn
import tensorflow as tf
import numpy as np
import pandas as pd
import os
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt

import tensorflow as tf
import tensorflow

from tensorflow import keras
from keras.layers import Dense

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
# We create PCA for Review Data and reduce dimensions-Training Data
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler#Not needed as there is not much difference i
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
import pandas
import math
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow/python/compat/
Instructions for updating:
non-resource variables are not supported in the long term
```

```
import numpy as np
import json
import gzip
import pandas as pdn
from urllib.request import urlopen
import keras
```

```
import rpy2
import numpy as np
from datetime import datetime
from statsmodels.tsa.seasonal import seasonal_decompose
import seaborn as sns
from statsmodels.tsa.statespace.sarimax import SARIMAX
!pip install statsmodels
from statsmodels.tsa.seasonal import seasonal_decompose
import seaborn as sns
from statsmodels.tsa.statespace.sarimax import SARIMAX
from collections import defaultdict
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (0
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: scipy>=0.18 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!unzip "/content/drive/My Drive/Wisconsin_Project/superstore_dataset2011-2015.csv.zip"
```

```
Archive:  /content/drive/My Drive/Wisconsin_Project/superstore_dataset2011-2015.csv.zip
  inflating: superstore_dataset2011-2015.csv
```

```
!unzip "/content/drive/My Drive/Wisconsin_Project/Combined_News_DJIA.csv.zip"
```

```
Archive:  /content/drive/My Drive/Wisconsin_Project/Combined_News_DJIA.csv.zip
  inflating: Combined_News_DJIA.csv
```

```
import pandas as pd
data = pd.read_csv('superstore_dataset2011-2015.csv', encoding= 'unicode_escape')
external_news=pd.read_csv('Combined_News_DJIA.csv')
```

```
data.iloc[0:20067,2]=pd.to_datetime(data.iloc[0:20067,2], format='%m/%d/%Y')#Y REPRESRENTS YE
```

```
data.iloc[20068:,2]=pd.to_datetime(data.iloc[20068:,2], format='%d-%m-%Y')#Y
```

Double-click (or enter) to edit

```
data['year'] = pd.to_datetime(data['Order Date']).dt.year


def function(df):
  df['year']= pd.to_datetime(df['Order Date']).dt.year
  df['month']= pd.to_datetime(df['Order Date']).dt.month
  df['day']= pd.to_datetime(df['Order Date']).dt.day
  return df


data_features=function(data)


data_features.shape
```

```
(51290, 27)
```

```
#Creating a String and coverting into data time object and then extracting date
validation_date = pd.to_datetime('2014-06-25')
data['Order Date']=pd.to_datetime(data['Order Date'])
validation_data = data.loc[data['Order Date']> validation_date]


start_date='2011-01-01'
end_training_date='2014-06-25'
end_date='2014-12-31'
start=pd.to_datetime(start_date)
end_tr=pd.to_datetime(end_training_date)
end_d=pd.to_datetime(end_date)
external_news_project=external_news[(external_news['Date']>start_date) &(external_news['Date'


external_news_project.head(2)
external_news_project.drop(["Label"], axis = 1, inplace = True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4913: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  errors=errors,
```

```
external_news_project.shape
```

```
(874, 26)
```

The above represents the news text items

External News Data

```
external_news_project_valid=external_news[external_news['Date']>end_training_date ]
external_news_project_valid2=external_news_project_valid.drop(["Label"], axis = 1, inplace =
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4913: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  errors=errors,
```

```
external_news_validation=external_news[(external_news['Date']>end_training_date) & (external_
```

```
external_news_validation.head(1)
```

```
external_news_validation2=external_news_validation.drop(["Label"], axis = 1, inplace = True)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:4913: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  errors=errors,
```

```
external_news_project_test=external_news[(external_news['Date']>='2015-01-01') &(external_new
#external_news_project_test=external_news_project_test.drop(["Label"], axis = 1, inplace = Tr
```

```
external_news_project['combined']=external_news_project[['Top1', 'Top2','Top3','Top4','Top5',
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  """Entry point for launching an IPython kernel.
```

```
top_headings=['Top1', 'Top2','Top3','Top4','Top5','Top6','Top7','Top8','Top9','Top10','Top11'
```

```
external_news_project.head(4)
```

```
external_news_validation['combined']=external_news_validation[['Top1', 'Top2','Top3','Top4','
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  """Entry point for launching an IPython kernel.
```

```
external_news_project_test['Combined']=external_news_project_test[['Top1', 'Top2','Top3','Top
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  """Entry point for launching an IPython kernel.
```

```
external_news_project_test['Combined']
```

```
1611    Most cases of cancer are the result of sheer b...
1612    Moscow-&gt;Beijing high speed train will reduc...
1613    US oil falls below $50 a barrel,Toyota gives a...
1614    'Shots fired' at French magazine HQ,90% of Bib...
1615    New Charlie Hebdo issue to come out next week:...
                              ...
1668    Germanwings Pilot Was Locked Out of Cockpit Be...
1669    WikiLeaks Reveals TPP Proposal Allowing Corpor...
1670    The president and CEO of The Associated Press ...
1671    Facebook 'tracks all visitors, breaching EU la...
1672    Indian Army team heads for Mt.Everest to bring...
Name: Combined, Length: 62, dtype: object
```

```
external_news_validation['combined']
```

```
1480    U.S. Scientist Offers $10,000 to Anyone Who Ca...
1481    German government cancels Verizon contract in ...
1482    Blackwaters top manager issued a threat: that ...
1483    Ukraine president ends ceasefire - 'We will at...
1484    Facebook Is Under Investigation For Mood Manip...
                              ...
1606    Death toll among Qatars 2022 World Cup workers...
1607    Saudis are eagerly awaiting the approval of a ...
1608    Solar Power Storage Prices Drop 25% In Germany...
1609    China businessman jailed for 13 years for buyi...
1610    AirAsia flight found at the bottom of the Java...
Name: combined, Length: 131, dtype: object
```

```
!pip install texthero
```

```
Collecting texthero
```

```
  Downloading texthero-1.1.0-py3-none-any.whl (24 kB)
Requirement already satisfied: wordcloud>=1.5.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-p
Collecting nltk>=3.3
  Downloading nltk-3.7-py3-none-any.whl (1.5 MB)
     |████████████████████████████████| 1.5 MB 3.2 MB/s
Collecting unidecode>=1.1.1
  Downloading Unidecode-1.3.4-py3-none-any.whl (235 kB)
     |████████████████████████████████| 235 kB 5.9 MB/s
Requirement already satisfied: spacy<3.0.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: gensim<4.0,>=3.6.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: tqdm>=4.3 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: matplotlib>=3.1.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: plotly>=4.2.0 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pandas>=1.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/loca
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-pa
Collecting regex>=2021.8.3
  Downloading regex-2022.3.15-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.
     |████████████████████████████████| 749 kB 6.6 MB/s
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: wasabi<1.1.0,>=0.4.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: blis<0.5.0,>=0.4.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: plac<1.2.0,>=0.9.6 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: srsly<1.1.0,>=1.0.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: catalogue<1.1.0,>=0.0.7 in /usr/local/lib/python3.7/d
Requirement already satisfied: thinc==7.4.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7
Requirement already satisfied: importlib-metadata>=0.20 in /usr/local/lib/python3.7/
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: pillow in /usr/local/lib/python3.7/dist-packages (fro
Installing collected packages: regex, unidecode, nltk, texthero
  Attempting uninstall: regex
    Found existing installation: regex 2019.12.20
    Uninstalling regex-2019.12.20:
      Successfully uninstalled regex-2019.12.20
```

```
#Now we need to improve the data Quality by pre processing it by improving the corpus.
#Towards this the text hero pipe line was easy to implement

def cleantext(textcolumn):
 import texthero as hero
 from texthero import preprocessing
 import re
 X=textcolumn
 custom_pipeline = [preprocessing.fillna,
                    preprocessing.lowercase,
                    preprocessing.remove_whitespace,
                    preprocessing.remove_diacritics,
                    preprocessing.remove_punctuation,
                    preprocessing.remove_brackets,
                    preprocessing.remove_stopwords,
                    preprocessing.remove_digits]
 Xm= hero.clean(X, custom_pipeline)
 Xm= [n.replace('{','') for n in Xm]
 Xm= [n.replace('}','') for n in Xm]
 Xm =[n.replace('(','') for n in Xm]
 Xm= [n.replace(')','') for n in Xm]
 Xm=[re.sub(r"[A-Za-z]+\d+|\d+[A-Za-z]+",'',i).strip() for i in Xm]#removing words with numbe
 return Xm


#corpus=external_news['combined']


train_external_news=cleantext(external_news_project.combined)
valid_external_news=cleantext(external_news_validation.combined)
test_external_news=cleantext(external_news_project_test['Combined'])
```

```
     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Unzipping corpora/stopwords.zip.
```

TFIDF VECTOR

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

def TFIDF_text(cleanedcorpusdf):
    countvectorizer = CountVectorizer(analyzer= 'word', stop_words='english')
    tfidfvectorizer = TfidfVectorizer(analyzer='word',stop_words= 'english')
    # convert the reviews  into a matrix
    count_wm = countvectorizer.fit_transform(cleanedcorpusdf)# this creates a list of review
```

```
    tfidf_wm = tfidfvectorizer.fit_transform(cleanedcorpusdf)# this creates a  matrix of TFID
    print(count_wm.toarray())
    #retrieve the terms found in the corpora
    # if we take same parameters on both Classes(CountVectorizer and TfidfVectorizer) , it wi
    #count_tokens = tfidfvectorizer.get_feature_names() # no difference
    count_tokens = countvectorizer.get_feature_names()# Feaures/ Words names
    tfidf_tokens = tfidfvectorizer.get_feature_names()
    df_countvect = pd.DataFrame(data = count_wm.toarray(),columns = count_tokens)
    df_tfidfvect = pd.DataFrame(data = tfidf_wm.toarray(),columns = tfidf_tokens)

    return df_tfidfvect,df_countvect

tf_idf_train=TFIDF_text(train_external_news)[0]
tf_idf_valid=TFIDF_text(valid_external_news)[0]
tf_idf_test=TFIDF_text(test_external_news)[0]
```

```
 [[0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  ...
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]
  [0 0 0 ... 0 0 0]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

```
tf_idf_train.shape
```

```
(874, 17016)
```

```
validation_date = pd.to_datetime('2014-06-25').date()
data['Order Date']=pd.to_datetime(data['Order Date'])
```

```
validation_data=data[data['Order Date']> '2014-06-25']
```

```
training_data = data.drop(labels=validation_data.index, axis=0)
consumer_vector=['Region',"Market","Country","Category",'Order Date']
```

```
training_data['Order Date']=pd.to_datetime(training_data['Order Date'])
```

```
training_data2=training_data.groupby(consumer_vector)['Sales'].sum().reset_index()
```

```
training_data2['Order Date'].dtype
```

```
dtype('<M8[ns]')
```

```
training_data_cohort=training_data2.set_index('Order Date')
```

```
Consumer_Vector=['Region','Market','Country','Category']
```

```
#Let us first take a sample data:'West',US,Tech
```

```
training_data_cohort=training_data2[(training_data2['Category']=='Technology') & (training_da
```

```
training_data_cohort.shape
```

```
(1052, 6)
```

```
training_data_cohort2=training_data_cohort.set_index('Order Date')
```

```
training_data_cohort2
```

```
del training_data_cohort2['Region']
del training_data_cohort2['Country']
del training_data_cohort2['Category']


del training_data_cohort2['Market']


training_data_cohort3=training_data_cohort2.resample('D').sum()


training_data_cohort3
```

```
training_data_cohort3[training_data_cohort3['Sales']==0].count()
```

```
Sales    571
dtype: int64
```

```
training_data_cohort_all_products_us=training_data2[(training_data2['Country']=='United State
```

```
training_data_us_all=training_data_cohort_all_products_us.set_index('Order Date')
del training_data_us_all['Market']
del training_data_us_all['Country']
del training_data_us_all['Region']
del training_data_us_all['Category']
```

Daily Sales of All Products in United States

```
training_data_us_all
```

```
#Training _Data
```

```
training_data_us_all=training_data_us_all.resample('D').sum()
```

```
training_data_us_all.reset_index(inplace=True)
```

```
training_data_us_all
```

```
external_news_project2=external_news_project.rename(columns={'Date': 'Order Date'})
```

```
external_news_project2['Order Date']= pd.to_datetime(external_news_project2['Order Date'])
```

EXTERNAL NEWS DATA WITH TOTAL US DAILY SALES

```
external_news_project2.head(2)
```

```
# Data Set for US Market for all kind of Total Sales with External News Parameter
```

Combine News Data with Daily Total Sales of United States online Sales through inner merege with sales data

```
training_data.head(1)
```

Merging News Data with Total Daily Sales of United States for All products

```
train_news_sales=external_news_project2.merge(training_data_us_all,on='Order Date',how='inner
```

```
train_news_sales.head(3)
train_news_sales.columns
```

```
Index(['Order Date', 'Top1', 'Top2', 'Top3', 'Top4', 'Top5', 'Top6', 'Top7',
       'Top8', 'Top9', 'Top10', 'Top11', 'Top12', 'Top13', 'Top14', 'Top15',
       'Top16', 'Top17', 'Top18', 'Top19', 'Top20', 'Top21', 'Top22', 'Top23',
       'Top24', 'Top25', 'combined', 'Sales'],
      dtype='object')
```

```
train_news_sales.head(2)
```

```
news_train_daily=train_news_sales[top_headings].agg(lambda x: ','.join(x.values), axis=1).T
```

```
news_clean_train=cleantext(news_train_daily)
tfidf_news_train=TFIDF_text(news_clean_train)[0]
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

CREATING ORDER DATE ITEMS TO LIST WHICH CAN BE USES AS A COLUMN TO BE ADDED AS A DATAFRAME

```
r=train_news_sales['Order Date'].tolist()
```

```
tfidf_news_train
```

```
# We create PCA for Head Line News Data and reduce dimensions-Training Data
import matplotlib.pyplot as plt
pca = PCA(n_components=3)
p=tfidf_news_train#Train_data
l=pca.fit(p)
print(l.explained_variance_ratio_)
PC=pca.fit_transform(p)
principalDf = pd.DataFrame(data = PC, columns = ['principal component 1', 'principal componen
principalDf
#Very low variance  and hence may not be accurate.
pca=PCA()
pca.fit(p)
cumsum=np.cumsum(pca.explained_variance_ratio_)

plt.plot(cumsum)
plt.xlabel("Number of Eigen Vectors Needed ")
plt.ylabel("Explained Variance")
cumsum
```

```
#DF_Train_TFIDF=pd.concat([tfidf_news_train,train_news_sales['Sales']],axis=1)#Final  Data Fr


#DF_Train_TFIDF.shape


# We create PCA for Head Line News Data and reduce dimensions-Training Data
# import matplotlib.pyplot as plt
# pca = PCA(n_components=20)
# p=tfidf_news_train#Train_data nes only
# l=pca.fit(p)
# print(l.explained_variance_ratio_)
```

```
# PC=pca.fit_transform(p)
# principalDf = pd.DataFrame(data = PC, columns = ['principal component 1', 'principal compon
# principalDf
# #Very low variance  and hence may not be accurate.
# pca=PCA()
# pca.fit(p)
# cumsum=np.cumsum(pca.explained_variance_ratio_)

# plt.plot(cumsum)
# plt.xlabel("Number of Eigen Vectors Needed ")
# plt.ylabel("Explained Variance")



%timeit
pca = PCA(n_components=500)
p=tfidf_news_train#Train_data
l=pca.fit(p)
print(l.explained_variance_ratio_)
PC=pca.fit_transform(p)
principalDf = pd.DataFrame(data = PC)
```

```
[0.00504171 0.00444821 0.00424299 0.00352573 0.00315856 0.00302291
 0.00292333 0.0028696  0.0027745  0.00265093 0.002594   0.00253464
 0.00250888 0.00248148 0.00243505 0.00242515 0.00238142 0.00236482
 0.00233661 0.00230861 0.00230739 0.00228746 0.00228096 0.00226068
 0.00225627 0.00221371 0.00220053 0.00219655 0.00219356 0.00217243
 0.00216199 0.0021481  0.00213191 0.00212319 0.00211419 0.00209847
 0.00208945 0.00208292 0.00207274 0.00205452 0.00205272 0.00204453
 0.00203348 0.00202701 0.00201379 0.00200451 0.0019984  0.00198846
 0.0019776  0.00197154 0.0019696  0.00196236 0.00195563 0.00194672
 0.00194212 0.00193899 0.0019369  0.00192205 0.00192137 0.00191117
 0.00190832 0.00190209 0.00189537 0.00189293 0.00188838 0.00188358
 0.00187589 0.00186846 0.00186488 0.00185896 0.00185835 0.0018508
 0.00184248 0.0018417  0.00183453 0.00183052 0.00182361 0.00181859
 0.00180926 0.00180854 0.00180328 0.00180075 0.00179519 0.00178733
 0.00178557 0.00178467 0.00177967 0.00177453 0.00177312 0.00176411
 0.00175811 0.0017571  0.00174734 0.00174329 0.00173927 0.00173551
 0.00173451 0.00173154 0.00172597 0.00172325 0.00171912 0.00171461
 0.0017131  0.00170947 0.00170788 0.00170244 0.0016991  0.00169617
 0.00169064 0.00168825 0.00168408 0.00168089 0.00167722 0.00167633
 0.00167253 0.00166827 0.00166492 0.00166235 0.00165774 0.0016525
 0.00165091 0.00164939 0.00164289 0.00164061 0.00163851 0.00163496
 0.00162916 0.00162446 0.00162257 0.00162027 0.00161694 0.00161385
 0.00160754 0.00160552 0.00160356 0.00159657 0.0015946  0.00158986
 0.00158855 0.00158838 0.00158488 0.00158005 0.00157858 0.00157711
 0.00157328 0.00156879 0.00156552 0.0015633  0.0015576  0.00155542
 0.00155281 0.00155118 0.0015473  0.00154465 0.00154404 0.00153906
 0.0015352  0.00153252 0.00152805 0.00152713 0.00152301 0.00152166
 0.00152023 0.0015167  0.0015135  0.00151006 0.00150838 0.00150566
 0.00150083 0.00149997 0.00149621 0.00149584 0.0014947  0.00148857
 0.00148657 0.00148379 0.00147899 0.00147632 0.00147368 0.00147345
 0.00147086 0.00146788 0.00146591 0.00146294 0.00146132 0.00145708
 0.00145487 0.00145198 0.00145057 0.0014479  0.00144585 0.00144376
```

```
       0.00144221 0.00144092 0.00143738 0.00143263 0.00143095 0.00142954
       0.0014281  0.00142577 0.00142458 0.00142125 0.00141804 0.00141649
       0.00141372 0.00141195 0.0014097  0.00140713 0.00140445 0.0014025
       0.00139954 0.0013958  0.00139487 0.00139127 0.00138956 0.00138805
       0.00138681 0.00138161 0.0013806  0.00137841 0.0013764  0.00137528
       0.00137343 0.00137115 0.00136888 0.00136529 0.00136392 0.00136239
       0.00136076 0.00135687 0.00135521 0.00135161 0.00135016 0.00134613
       0.00134595 0.00134242 0.00134225 0.00133954 0.00133745 0.00133655
       0.00133371 0.00133077 0.00132883 0.0013267  0.00132362 0.00132144
       0.00132085 0.0013199  0.00131839 0.0013146  0.00131314 0.00131246
       0.00130946 0.00130835 0.00130455 0.0013023  0.00130107 0.00129926
       0.00129618 0.00129154 0.00128994 0.00128954 0.00128571 0.0012852
       0.00128486 0.0012832  0.00128243 0.00127707 0.0012744  0.0012729
       0.00127234 0.00126964 0.00126654 0.00126472 0.00126282 0.00126108
       0.00126041 0.00125787 0.00125573 0.00125226 0.00125113 0.00124982
       0.00124633 0.00124487 0.00124335 0.00124252 0.00123896 0.00123775
       0.00123523 0.00123404 0.00123314 0.0012319  0.00122856 0.00122702
       0.00122537 0.00122294 0.00122128 0.00121783 0.00121607 0.00121476
       0.00121392 0.00121142 0.00120868 0.00120729 0.00120526 0.00120262
       0.00120138 0.00119697 0.00119576 0.00119319 0.00119202 0.00119133
       0.00118942 0.00118891 0.00118528 0.00118445 0.00118241 0.00118102
       0.00117912 0.0011769  0.00117346 0.00117238 0.0011706  0.00116948
       0.00116768 0.0011657  0.0011641  0.00116245 0.0011608  0.00115934
       0.00115637 0.00115551 0.00115359 0.00115126 0.00115048 0.00114826
       0.00114625 0.00114547 0.00114266 0.00114087 0.00113759 0.00113494
       0.00113283 0.00113164 0.00113014 0.00112812 0.00112612 0.0011255
```

```
principalDf.head(3)
```

```
DF_Train_TFIDF_sales=pd.concat([principalDf,train_news_sales['Sales']],axis=1)
```

```
DF_Train_TFIDF_sales.head(2)
```

```
train_news_sales.head(1)
```

```
train_news_sales.shape
```

```
(874, 28)
```

Adding Order Date

```
DF_Train_TFIDF_sales['Order Date']=r
```

Section I complete

```
DF_Train_TFIDF_sales.head(2)
```

TRAINING DATA _TFIDF FRAMEWORK

```
Ready_X_TFIDF=function(DF_Train_TFIDF_sales)
```

```
Ready_X_TFIDF.head(3)
```

```
!pip install gensim
train_external_news# After Hero Processsing
from gensim.models.doc2vec import Doc2Vec,TaggedDocument
documents=[TaggedDocument(p,[i]) for i ,p in enumerate(train_external_news)]
documents[1:10]# each document  is like a post

def parameter_tuning(v,count):
    #instantiate model
    model = Doc2Vec(vector_size=v, window=3,min_count=count , workers=4, epochs = 3)#build vo
    model.build_vocab(documents)#train model
    model.train(documents, epochs=model.epochs,total_examples=model.corpus_count)
    return model

x=parameter_tuning(v=16,count=40)# Parametric tuning of #model with vector of 32, count 40
x2=parameter_tuning(v=8,count=30)
x3=parameter_tuning(v=32,count=20)
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packag
```

```
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist-packages (
```

```
def doc2vec(cleanedcorpus_train_ADHD):

  doc2vecnew = [x.infer_vector((cleanedcorpus_train_ADHD[i].split(' ')))
              for i in range(0,len(cleanedcorpus_train_ADHD))]
  doc2vecnew2=[x2.infer_vector((cleanedcorpus_train_ADHD[i].split(' ')))
              for i in range(0,len(cleanedcorpus_train_ADHD))]
  doc2vecnew3=[x3.infer_vector((cleanedcorpus_train_ADHD[i].split(' ')))
              for i in range(0,len(cleanedcorpus_train_ADHD))]
  # Each document is vector in R 64 space

  X16_train=pd.DataFrame(doc2vecnew)
  X8_train=pd.DataFrame(doc2vecnew2)
  X32_train=pd.DataFrame(doc2vecnew3)

  return X16_train,X8_train,X32_train
  # Each document is vector in R 64 space
```

Let us take US consumer all products daily sales to start with

```
text_us_allprod=cleantext(train_news_sales['combined'])
```

```
p,q,t=doc2vec(text_us_allprod)
```

```
p
```

```
p.shape
```

```
(874, 16)
```

```
p.head(3)
```

```
from datetime import datetime
def parse(row):
    x=row['Order Date']
    return datetime.strptime(x, '%Y %m %d %H')
```

```
data_features.columns
```

```
Index(['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode',
       'Customer ID', 'Customer Name', 'Segment', 'City', 'State', 'Country',
       'Postal Code', 'Market', 'Region', 'Product ID', 'Category',
       'Sub-Category', 'Product Name', 'Sales', 'Quantity', 'Discount',
       'Profit', 'Shipping Cost', 'Order Priority', 'year', 'month', 'day'],
      dtype='object')
```

```
data_features['Order Date']=pd.to_datetime(data_features['Order Date'],errors='coerce')
```

```
data_features['Order Date'].dtype
```

```
dtype('<M8[ns]')
```

```
DF_Doc2Vec_train_sales=pd.concat([p,train_news_sales['Sales']],axis=1)
DF_Doc2Vec_train_sales.head(2)
```

```
train_news_sales['Order Date']=r
```

```
r[0]
```

```
Timestamp('2011-01-03 00:00:00')
```

```
DF_Doc2Vec_train_sales['Order Date']=r
DF_Doc2Vec_train_sales.head(2)
```

```
DF_Doc2Vec_train_sales
```

```
x_train_doc2vec=DF_Doc2Vec_train_sales.iloc[:,:-1]
```

```
x_train_doc2vec
```

SECTON 2 FINISHES OF PREPARING doc2 Vector for Training Data Set

```
from sklearn.preprocessing import MinMaxScaler

values = x_train_doc2vec.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
```

```
values.shape
```

```
(874, 17)
```

```
values[:,-1]
```

```
array([2.2031510e+03, 1.1988800e+02, 0.0000000e+00, 5.1885200e+03,
       6.0102400e+02, 4.7100000e+00, 4.7320800e+03, 5.6243900e+03,
       0.0000000e+00, 3.5537950e+03, 0.0000000e+00, 6.4864000e+01,
       3.7859400e+02, 2.6738700e+03, 4.6020000e+01, 0.0000000e+00,
       0.0000000e+00, 1.0972500e+03, 4.2667000e+02, 2.4050000e+02,
       0.0000000e+00, 4.6890000e+02, 2.0238400e+02, 1.4585580e+03,
       7.9560000e+01, 8.8628000e+02, 3.2541500e+03, 5.9814400e+02,
       2.1263700e+03, 0.0000000e+00, 5.7672600e+02, 2.1360000e+01,
       9.0400000e+00, 5.4208000e+01, 8.8500000e+00, 1.9440000e+01,
       1.1364000e+01, 5.5672000e+01, 1.9456000e+01, 0.0000000e+00,
       2.1164600e+02, 1.3458920e+03, 2.3345000e+02, 0.0000000e+00,
       4.7799400e+02, 2.2080000e+01, 8.6268300e+02, 2.7841630e+03,
       2.1085500e+03, 3.7078200e+02, 4.7192000e+02, 3.9603580e+03,
       2.8106716e+04, 4.1098160e+03, 4.6409300e+02, 9.4506400e+02,
       6.5380000e+01, 4.5914600e+02, 1.8238040e+03, 8.9084100e+02,
       1.1703220e+03, 1.9595520e+03, 1.6448000e+01, 4.7584400e+02,
       1.8743200e+02, 4.5627200e+02, 2.4740800e+02, 2.7431540e+03,
       6.6937480e+03, 1.2998000e+02, 6.3857600e+02, 0.0000000e+00,
       2.9472000e+02, 1.0205320e+03, 2.0547000e+02, 1.2504500e+03,
       8.4536000e+02, 2.3799940e+03, 2.8256800e+02, 0.0000000e+00,
       2.5784760e+03, 7.6884400e+02, 1.3438400e+02, 7.0556200e+02,
       5.2950980e+03, 2.6097700e+02, 0.0000000e+00, 2.9630400e+02,
       1.9190000e+02, 1.6945640e+03, 2.9636420e+03, 8.9838200e+02,
       2.8928000e+02, 9.1680000e+01, 9.4894800e+02, 9.1620000e+01,
       3.2010400e+02, 3.2375800e+02, 1.1628000e+02, 1.1954000e+02,
       1.3257180e+03, 2.1573940e+03, 7.7370000e+02, 1.9536000e+01,
       0.0000000e+00, 0.0000000e+00, 1.3740010e+03, 1.0384720e+03,
       2.6289000e+02, 1.0036000e+02, 4.9471000e+02, 1.4520000e+01,
       2.1294000e+02, 9.4297400e+02, 7.6379200e+02, 3.4185020e+03,
       1.8289496e+03, 4.1074430e+03, 1.9754980e+03, 1.0949600e+02,
       4.2720000e+00, 6.1614000e+02, 1.6457910e+03, 4.6680000e+01,
```

```
              7.3962400e+02, 4.4071000e+03, 1.0913500e+03, 2.8139700e+02,
              2.4119400e+02, 0.0000000e+00, 1.4491420e+03, 5.4840000e+02,
              3.5121600e+02, 2.9259400e+02, 9.5100000e+00, 2.7304000e+02,
              4.6132600e+02, 2.2857860e+03, 1.9614820e+03, 1.3483440e+03,
              8.3412900e+03, 5.0399940e+03, 5.8006000e+02, 1.9315200e+02,
              0.0000000e+00, 8.7158000e+01, 1.8032000e+02, 0.0000000e+00,
              1.9587500e+03, 1.7999700e+03, 3.9556290e+03, 1.4228428e+04,
              1.4739200e+02, 3.3357600e+02, 1.0951200e+03, 8.4278200e+02,
              8.5309200e+02, 5.0034200e+02, 0.0000000e+00, 1.4391060e+03,
              2.3104000e+01, 5.6522400e+02, 5.9740000e+01, 1.4516500e+03,
              3.6681000e+02, 8.3232250e+02, 1.2124000e+02, 9.2520000e+01,
              0.0000000e+00, 1.4560000e+01, 5.4630080e+03, 2.6552100e+02,
              3.6056850e+03, 4.0435880e+03, 1.5226520e+03, 1.2754080e+03,
              5.0237160e+03, 1.3343600e+02, 3.3552000e+01, 9.3386370e+03,
              5.3359680e+03, 2.6796520e+03, 1.2402660e+03, 1.0662337e+04,
              1.4806140e+03, 7.7344000e+02, 4.9029200e+02, 8.1090700e+03,
              9.8753200e+02, 7.4195600e+02, 0.0000000e+00, 2.3437990e+03,
              4.9155000e+02, 0.0000000e+00, 2.0493860e+03, 2.0025560e+03,
              5.7333600e+02, 2.2121820e+03, 4.5254840e+03, 1.3606800e+02,
              3.1349160e+03, 6.4110400e+02, 1.5011040e+03, 1.3566020e+03,
              5.6490000e+01, 1.0388200e+02, 4.4788000e+02, 2.2320000e+01,
              1.5105820e+03, 5.3620260e+03, 5.4830000e+01, 0.0000000e+00,
              9.1831400e+02, 1.9152400e+03, 2.1872060e+03, 4.5162000e+02,
              1.2795000e+02, 1.3811640e+03, 8.1145480e+03, 1.0347300e+03,
              1.7808220e+03, 1.3559285e+03, 1.1544274e+04, 3.4068550e+03,
              1.2496220e+03, 1.1032980e+03, 1.3632800e+03, 4.4156950e+03,
              2.2808300e+03, 6.4904400e+02, 2.1915400e+02, 9.9400000e+00,
```

```
scaler2=MinMaxScaler(feature_range=(0,1))
values2=values[:,-1].reshape(-1,1)
scaled2=scaler2.fit_transform(values2)
```

```
scaled.shape
```

```
     (874, 17)
```

```
len(scaled[0:-3,1])
```

```
     871
```

```
import numpy as np
SEQ_LEN = 4
DATA_LEN = scaled.shape[0]

X_train = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,17)#preparing training data from 0:-6

for i in range(1,SEQ_LEN):
    X_train = np.append(X_train, scaled[i:-SEQ_LEN+i-1,:].reshape(-1,1,17), axis=1)

Y_train = scaled[SEQ_LEN:-1,-1]# SEQUENCE OF TRAINED VALUES FROM
```

```
X_train.shape
```

```
(869, 4, 17)
```

```
Y_train.shape
```

```
(869,)
```

```
X_train.shape
```

```
(869, 4, 17)
```

```
X_train[1,0,0]
```

```
0.2378549713563315
```

```
#Lets chec if sequence is right or not
print(X_train[40,:,:17])# A SEQUENCE OF 40 ROW SHOULD HAVE ALL 5 POINTS 40-45
print(Y_train[40])
print(scaled[40:46,0])
```

```
[[0.26373641 0.43435967 0.66555879 0.63667968 0.33719624 0.0710506
  0.60390963 0.05930059 0.0808841  0.68908118 0.17250621 0.28174283
  0.25525936 0.15931665 0.75801327 0.14353107 0.00753009]
 [0.0615997  0.2454285  0.76215645 0.53136062 0.53869318 0.22534327
  0.39968923 0.35286944 0.18747113 0.4857868  0.08067802 0.34484592
  0.04608589 0.32150565 0.8524199  0.32467424 0.04788507]
 [0.29312938 0.38422627 0.78174548 0.06768063 0.47647984 0.14715629
  0.69523853 0.5659632  0.06526819 0.24758881 0.21330993 0.51954549
  0.02522123 0.41537503 0.79469731 0.05997214 0.00830584]
 [0.016608   0.22750482 0.85474119 0.46192216 0.28033616 0.09178523
  0.35209144 0.21027999 0.15116155 0.3350958  0.48355536 0.34117759
  0.26912566 0.04258915 0.81952845 0.22887015 0.        ]]
0.0170063980437985
[0.26373641 0.0615997  0.29312938 0.016608   0.1020476  0.07284773]
```

```
Y_train
```

```
array([2.13836437e-02, 1.67575607e-04, 1.68361185e-01, 2.00108401e-01,
       0.00000000e+00, 1.26439353e-01, 0.00000000e+00, 2.30777584e-03,
       1.34698767e-02, 9.51327789e-02, 1.63733109e-03, 0.00000000e+00,
       0.00000000e+00, 3.90387123e-02, 1.51803576e-02, 8.55667379e-03,
       0.00000000e+00, 1.66828455e-02, 7.20055662e-03, 5.18935759e-02,
       2.83064019e-03, 3.15326771e-02, 1.15778378e-01, 2.12811771e-02,
       7.56534488e-02, 0.00000000e+00, 2.05191528e-02, 7.59960715e-04,
```

```
3.21631314e-04, 1.92864937e-03, 3.14871364e-04, 6.91649640e-04,
4.04316178e-04, 1.98073656e-03, 6.92218899e-04, 0.00000000e+00,
7.53008640e-03, 4.78850678e-02, 8.30584406e-03, 0.00000000e+00,
1.70063980e-02, 7.85577369e-04, 3.06931269e-02, 9.90568589e-02,
7.50194366e-02, 1.31919360e-02, 1.67902931e-02, 1.40904330e-01,
1.00000000e+00, 1.46221850e-01, 1.65118187e-02, 3.36241345e-02,
2.32613444e-03, 1.63358110e-02, 6.48885483e-02, 3.16949515e-02,
4.16385180e-02, 6.97182837e-02, 5.85198214e-04, 1.69299039e-02,
6.66858412e-03, 1.62335578e-02, 8.80245134e-03, 9.75978126e-02,
2.38154753e-01, 4.62451750e-03, 2.27196945e-02, 0.00000000e+00,
1.04857501e-02, 3.63091867e-02, 7.31035244e-03, 4.44893669e-02,
3.00767973e-02, 8.46770573e-02, 1.00533979e-02, 0.00000000e+00,
9.17387858e-02, 2.73544586e-02, 4.78120603e-03, 2.51029683e-02,
1.88392625e-01, 9.28521852e-03, 0.00000000e+00, 1.05421067e-02,
6.82754969e-03, 6.02903591e-02, 1.05442486e-01, 3.19632503e-02,
1.02922020e-02, 3.26185386e-03, 3.37623221e-02, 3.25971914e-03,
1.13888794e-02, 1.15188840e-02, 4.13708951e-03, 4.25307603e-03,
4.71673034e-02, 7.67572419e-02, 2.75272287e-02, 6.95065194e-04,
0.00000000e+00, 0.00000000e+00, 4.88851490e-02, 3.69474684e-02,
9.35328055e-03, 3.57067684e-03, 1.76011313e-02, 5.16602509e-04,
7.57612522e-03, 3.35497751e-02, 2.71747151e-02, 1.21625806e-01,
6.50716220e-02, 1.46137421e-01, 7.02856214e-02, 3.89572371e-03,
1.51992143e-04, 2.19214511e-02, 5.85550799e-02, 1.66081302e-03,
2.63148494e-02, 1.56798823e-01, 3.88287981e-02, 1.00117353e-02,
8.58136539e-03, 0.00000000e+00, 5.15585670e-02, 1.95113510e-02,
1.24958035e-02, 1.04101098e-02, 3.38353296e-04, 9.71440420e-03,
1.64133725e-02, 8.13252605e-02, 6.97869506e-02, 4.79723067e-02,
2.96772131e-01, 1.79316360e-01, 2.06377721e-02, 6.87209420e-03,
0.00000000e+00, 3.10096704e-03, 6.41554851e-03, 0.00000000e+00,
6.96897496e-02, 6.40405660e-02, 1.40736079e-01, 5.06228760e-01,
5.24401357e-03, 1.18681955e-02, 3.89629297e-02, 2.99850755e-02,
3.03518917e-02, 1.78015105e-02, 0.00000000e+00, 5.12014993e-02,
8.22009942e-04, 2.01099268e-02, 2.12547065e-03, 5.16477983e-02,
1.30506175e-02, 2.96129402e-02, 4.31355979e-03, 3.29173995e-03,
0.00000000e+00, 5.18025656e-04, 1.94366642e-01, 9.44688807e-03,
1.28285531e-01, 1.43865544e-01, 5.41739561e-02, 4.53773397e-02,
1.78737210e-01, 4.74747744e-03, 1.19373605e-03, 3.32256426e-01,
1.89846726e-01, 9.53384949e-02, 4.41270336e-02, 3.79351931e-01,
5.26782994e-02, 2.75179783e-02, 1.74439447e-02, 2.88510049e-01,
3.51350901e-02, 2.63978189e-02, 0.00000000e+00, 8.33892867e-02,
1.74887027e-02, 0.00000000e+00, 7.29144593e-02, 7.12483095e-02,
2.03985410e-02, 7.87065269e-02, 1.61010771e-01, 4.84112054e-03,
1.11536189e-01, 2.28096374e-02, 5.34073066e-02, 4.82661155e-02,
2.00983993e-03, 3.69598497e-03, 1.59349815e-02, 7.94116253e-04,
5.37445214e-02, 1.90773835e-01, 1.95077931e-03, 0.00000000e+00,
3.26724047e-02, 6.81417210e-02, 7.78179137e-02, 1.60680458e-02,
4.55229277e-03, 4.91399991e-02, 2.88704949e-01, 3.68143329e-02,
6.33593053e-02, 4.82421532e-02, 4.10730090e-01, 1.21211421e-01,
4.44599077e-02, 3.92538922e-02, 4.85037099e-02, 1.57104622e-01,
8.11489325e-02, 2.30921321e-02, 7.79721117e-03, 3.53652131e-04,
7.27014853e-02, 2.61448545e-02, 0.00000000e+00, 6.97346499e-02,
```

```
from tensorflow.keras import Sequential
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
```

```
from keras.models import Sequential


model = Sequential()
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam',metrics = ['mae'])


print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                13600

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 13,651
Trainable params: 13,651
Non-trainable params: 0
_________________________________________________________________
None
```

```
X_train.shape
```

```
(869, 4, 17)
```

```
epochs = 100
validation_split = 0.1

history = model.fit(X_train, Y_train, batch_size=128,
          epochs=epochs,
          validation_split=validation_split)

import matplotlib.pyplot as plt

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'], label='training')
plt.plot(history.epoch,history.history['val_loss'], label='validation')
plt.title('loss')
plt.legend(loc='best')
```

```
# A = np.array([])
# OFFSET = 100
# LEN = 40
# for i in range(0,LEN):
#     A = np.append(A,model.predict(X_train[OFFSET+i].reshape(1,5,17)))
```

```
X_train[0].reshape(1,4,17)
```

```
array([[[0.05034993, 0.27276439, 0.98729924, 0.50984203, 0.03917711,
         0.08894901, 0.26205375, 0.580655  , 0.11995784, 0.58551673,
         0.38306629, 0.76595918, 0.17376478, 0.20447118, 0.76616353,
         0.42030506, 0.07838522],
        [0.23785497, 0.53025431, 0.98723572, 0.22725861, 0.46130182,
         0.15204075, 0.2792846 , 0.46441378, 0.07541523, 0.92962932,
         0.57494168, 0.70957814, 0.23582264, 0.2188143 , 0.87436413,
         0.5104899 , 0.00426546],
        [0.13467945, 0.5152548 , 0.57667461, 0.12228095, 0.54091738,
         0.35771934, 0.54488903, 0.433683  , 0.31940881, 0.79281209,
         0.34146111, 0.83765468, 0.32191332, 0.26620453, 0.86900757,
         0.27681139, 0.        ],
        [0.5649698 , 0.44453869, 0.62631251, 0.39504283, 0.71598856,
         0.73929561, 0.11026317, 0.55326939, 0.70985502, 0.40325431,
         0.54865069, 0.84782282, 0.63172277, 0.54143801, 0.29252665,
         0.45063591, 0.18460072]]])
```

```
A = np.array([])
OFFSET = 100
LEN = 40
for i in range(0,LEN):
    A = np.append(A,model.predict(X_train[OFFSET+i].reshape(1,4,17)))
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training_v1.py:2079: UserWarning: `M
  updates=self.state_updates,
```

```
A.shape
```

```
(40,)
```

```
plt.plot(Y_train[OFFSET:OFFSET+LEN],label="Measured")
plt.plot(A,label="Predicted")
plt.legend()
```

```
#y_new_inverse = scaled2.inverse_transform(A)
```

```
A1=A.reshape(-1,1)
#_scaled = scaler.fit_transform(A1)
```

```
y_new_inverse = scaler2.inverse_transform(A1)
```

```
y_new_inverse.reshape(-1,)
```

```
array([1645.86172204, 1452.70414543, 1622.42848694, 1495.70917498,
       1591.75223668, 1320.18748957,  886.0137799 ,  972.6328072 ,
        938.39645923, 1213.15724241, 1314.69243097, 1405.37026911,
       1476.68310651, 1723.87415214, 1404.61261883, 1953.3153868 ,
       1600.22617178, 1700.63252844, 1608.3148947 , 1003.9167692 ,
        598.56183109, 1699.57699052, 1510.52921642, 1451.84901417,
       1235.96161521, 1609.55167815, 1388.24240985, 1626.38290623,
        652.19207856,  637.34654129,  735.20687471, 1237.56403088,
       1154.12638086, 1654.44235222, 2330.04368827, 1806.30024063,
       1789.37749979, 2101.12755258, 2158.93513772, 2247.9443862 ])
```

```
plt.plot(y_new_inverse)
plt.title("Forecasted value for 40 observations")
```

```
external_news_validation
```

```
from pandas.core.groupby import groupby
vect_gp=['Segment','Market']
```

```
 object2=data_features.groupby(vect_gp)
```

```
data_features.groupby(vect_gp)['Order Date'].count().reset_index().rename(columns={'Order Dat
```

```
Market_List=data_features['Market'].unique()
Category_List=data_features['Segment'].unique()
```

```
Market_List
Category_List
```

```
array(['Consumer', 'Home Office', 'Corporate'], dtype=object)
```

```
import numpy as np
from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')

imp_mean
```

```
SimpleImputer()
```

```
df=data_features
```

```
df=object2.get_group(('Consumer','Africa'))
df=df.loc[:,['Order Date','Sales']]
df.set_index('Order Date')
```

```
df2=df.set_index('Order Date')
```

```
df_m=df2.resample('d').sum()
```

```
df_m
```

```
df_mod_africa_consumer = df_m.reset_index()
```

```
df_mod_africa_consumer
```

```
sales=df_m['Sales']
df_m['Sales']=sales.replace(0, np.nan)
```

```
df_m
```

```
imputed=np.where((np.isnan(df_m['Sales'])),df_m['Sales'].mean(skipna=True),df_m['Sales'])
```

```
imputed
```

```
array([ 408.3      ,  597.6964866,  597.6964866, ...,  714.78     ,
       1603.128    ,  153.633    ])
```

```
df_m['Imputed_Sales']=imputed.tolist()
```

```
df_m
```

```
del df_m['Sales']
```

```
df_consumer_africa=df_m
```

```
# imp_mean.fit(df_m)
# df3=imp_mean.transform(df_m)
# df3
```

```
# df3.reshape(-1,)
```

```
# Market_List
# Category_List
```

```
df['Order Date']=pd.to_datetime(df['Order Date'],errors='coerce')
```

```
# df=object2.get_group((Category_List[0],Market_List[0]))
# df['Order Date']=pd.to_datetime(df['Order Date'],errors='coerce')
```

```
# df=df.loc[:,['Order Date','Sales']]
# df.set_index('Order Date')
```

```
# for i in  range(len(Market_List)):
#      for j  in range(len(Category_List)):
#          df=object2.get_group((Category_List[j],Market_List[i]))
#          df=df.loc[:,['Order Date','Sales']]
#          df.set_index('Order Date')
#          df.resample('d').mean()
#          imputed=np.where((np.isnan(df['Sales'])),df['Sales'].mean(skipna=True),df['Sales']
#          df.set_index('Order Date')
#          df['Imputed_Sales']=imputed.tolist()
#          del df['Sales']
```

```
data_ext_train=pd.concat([data_features,p],axis=1)
```

```
data_ext_train_africa=pd.concat([df_mod_africa_consumer,p],axis=1)
```

```
data_ext_train_africa
```

```
data_ext_train.tail(3)
data_ext_train2=data_ext_train.rename(columns={0:'VECT-1',1:'VECT-1',2:'VECT-2',3:'VECT-3',4:
```

```
column_names=[['Order Date','year','month','day','Market','Category','Sales','0','1','2','3',
```

```
vector=['Market','Category','Sales','Order Date','year','month','day','VECT-1','VECT-2','VECT
```

```
data_ext_train3=data_ext_train2[vector]
```

```
# Develop Neural Network Model
```

```
data_ext_train3[data_ext_train3['Market']=='US']
```

Preparing Validation Data/Inference Data for All Products in US

```
validation_data.head(1)
```

```
validation_short=validation_data[validation_data['Market']=='US'][['Sales','Order Date']]
```

```
validation_short2=validation_short.set_index(['Order Date']).resample('d').sum()
```

```
validation_short2.reset_index(inplace=True)
```

```
validation_short2
```

```
external_news_validation.head(1)
external_news_validation2=external_news_validation.rename(columns={'Date':'Order Date'})
```

```
clean_valid_corpus=cleantext(external_news_validation2['combined'])
```

```
s,u,v=doc2vec(clean_valid_corpus)
```

```
s.head(1)
```

```
s.shape
```

```
(131, 16)
```

```
external_news_validation2.head(1)
```

```
external_news_validation2['Order Date']=pd.to_datetime(external_news_validation2['Order Date'
```

```
# reate Order Date Here
```

```
# Create Sales here
```

```
validation_set=validation_short2.merge(external_news_validation2,on='Order Date',how='left')
```

```
validation_set['combined']=validation_set['combined'].fillna('No News')
```

```
text=validation_set['combined']
```

```
def doc2vec_gen(text):
   new=cleantext(text)#cleaning text
   f,g,h=doc2vec(new)
   return f
```

```
valid_vector=doc2vec_gen(text)
```

```
valid_vector['Sales']=validation_short2['Sales']
```

```
valid_vector
```

```
mean1=valid_vector['Sales'].mean()

valid_vector['Sales']=valid_vector['Sales'].replace(0.00,mean1)#replacing 0 values

valid_vector.head(10)
```

```
valid_vector2=valid_vector.iloc[:-9,:]
```

```
valid_vector2
```

```
def scaling(X):
  value_1= X.values#np.array
  scalar=MinMaxScaler(feature_range=(0,1))
  scaled=scaler.fit_transform(value_1)
  return scaled
```

```
valid_vector
```

```
valid_vector2=valid_vector.iloc[:-9,]# to create a 3 dimensional matrix
valid_vector2.shape
```

```
(180, 17)
```

```
y_actual=valid_vector2.iloc[4:,-1]
y_actual
```

```
4      4521.9912
5        33.7400
6       904.3540
7       778.2360
8       114.4200
          ...
175    2027.7580
176    3645.9110
177    1895.9260
178     377.7360
179    2140.9400
Name: Sales, Length: 176, dtype: float64
```

```
valid_vector_uni=valid_vector2.iloc[:,16]
```

```
valid_vector_uni
```

```
0      1508.352000
1      4224.136000
2      1644.906000
3      2244.414278
4      4521.991200
           ...
175    2027.758000
176    3645.911000
177    1895.926000
178     377.736000
179    2140.940000
Name: Sales, Length: 180, dtype: float64
```

```
#y_ACT.shape


#actual=y_ACT[:,0]


#print(len(actual))


valid_vector2.shape
```

```
(180, 17)
```

```
def transform(X):
   val2=X.loc['Sales'].values
   val2.reshape(-1,1)
   scalar2=MinMaxScaler(feature_range=(0,1))
   scaled2=scaler.fit(transform(val2))
   y_train=scaler2.inverse_transform(val2)
   return y_train


scaled_valid=scaling(valid_vector2)


scaled_valid.shape
```

```
(180, 17)
```

```
#testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
X_test=scaled_valid[0:-SEQ_LEN-1,:].reshape(-1,1,17)



for i in range(1,SEQ_LEN):
    X_test = np.append(X_test, scaled_valid[i:-SEQ_LEN+i-1,:].reshape(-1,1,17), axis=1)


X_test.shape
```

```
(175, 4, 17)
```

```
yhat=model.predict(X_test)
```

```
#valid_3d=scaled_valid.reshape(1,4,17)
```

```
actual=valid_vector2['Sales'].values
```

```
print(len(actual))
```

```
180
```

```
values2=actual.reshape(-1,1)
```

```
values2
```

```
array([[1.50835200e+03],
       [4.22413600e+03],
       [1.64490600e+03],
       [2.24441428e+03],
       [4.52199120e+03],
       [3.37400000e+01],
       [9.04354000e+02],
       [7.78236000e+02],
       [1.14420000e+02],
       [3.18336980e+03],
       [3.13500000e+01],
       [3.80378000e+02],
       [1.42402600e+03],
       [2.24441428e+03],
       [1.49965200e+03],
```

```
            [4.28875000e+03],
            [1.06800000e+01],
            [3.81600000e+00],
            [5.04542000e+02],
            [3.59893400e+03],
            [2.13222900e+03],
            [3.16636000e+02],
            [2.19187300e+03],
            [2.10679400e+03],
            [2.24441428e+03],
            [2.28320800e+03],
            [3.68594400e+03],
            [5.56314000e+02],
            [8.98188000e+02],
            [2.39996000e+03],
            [7.98742000e+02],
            [2.01230200e+03],
            [1.14268000e+03],
            [1.17056400e+03],
            [1.39129400e+03],
            [5.23376000e+02],
            [3.39559000e+03],
            [2.24441428e+03],
            [1.66590000e+02],
            [1.97129050e+03],
            [2.54946800e+03],
            [2.24441428e+03],
            [1.62025000e+03],
            [3.47962400e+03],
            [3.84856500e+03],
            [2.74921000e+03],
            [2.41337800e+03],
            [2.91651400e+03],
            [3.69308400e+03],
            [2.64316400e+03],
            [4.40736000e+02],
            [1.94987200e+03],
            [8.05640000e+01],
            [9.51728800e+03],
            [7.07848400e+03],
            [5.12446000e+02],
            [6.92950000e+02],
```

```
scaler2=MinMaxScaler(feature_range=(0,1))
scaled2=scaler2.fit_transform(values2)
#inverse_number=scaled2.inverse_transform(yhat)
```

```
yhat_unscaled=scaler2.inverse_transform(yhat)
```

```
yhat_unscaled
```

```
     array([[1107.839  ],
```

```
[1250.4471 ],
[ 863.9075 ],
[ 961.96533],
[1338.0143 ],
[ 853.66174],
[ 859.8684 ],
[ 485.34448],
[ 377.33978],
[ 445.72757],
[ 577.41974],
[ 647.1608 ],
[1258.3226 ],
[1155.8228 ],
[ 977.0435 ],
[ 771.04364],
[ 764.75867],
[1200.4645 ],
[1209.9    ],
[1028.7065 ],
[ 988.4475 ],
[1026.944  ],
[ 945.0848 ],
[ 825.0592 ],
[ 999.8675 ],
[1261.9893 ],
[1555.0297 ],
[1119.657  ],
[1013.9987 ],
[ 269.2663 ],
[ 348.31442],
[ 762.81415],
[ 572.79535],
[ 992.2617 ],
[1104.389  ],
[ 611.16   ],
[ 527.2481 ],
[ 933.5462 ],
[1129.8474 ],
[1291.7384 ],
[1258.145  ],
[1389.6775 ],
[1327.6411 ],
[1220.6648 ],
[ 833.171  ],
[1474.9319 ],
[1452.7163 ],
[1267.1683 ],
[1279.886  ],
[ 960.3336 ],
[1276.5159 ],
[1400.765  ],
[1677.8501 ],
[2042.5717 ],
[2124.2522 ],
[1355.1969 ],
[1442.7975 ],
```

```
def MAPE(Y_Predicted,Y_actual):
     mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
     return mape

def test_evaluation(predictions,valid):
   df=pd.DataFrame(predictions)
   yhat=df.values.reshape(-1,)
   mape=MAPE(yhat,valid)
   return mape


yhat_array_unscaled=yhat_unscaled.reshape(-1,)


yhat_array_unscaled

     array([1107.839  , 1250.4471 ,  863.9075 ,  961.96533, 1338.0143 ,
             853.66174,  859.8684 ,  485.34448,  377.33978,  445.72757,
             577.41974,  647.1608 , 1258.3226 , 1155.8228 ,  977.0435 ,
             771.04364,  764.75867, 1200.4645 , 1209.9    , 1028.7065 ,
             988.4475 , 1026.944  ,  945.0848 ,  825.0592 ,  999.8675 ,
            1261.9893 , 1555.0297 , 1119.657  , 1013.9987 ,  269.2663 ,
             348.31442,  762.81415,  572.79535,  992.2617 , 1104.389  ,
             611.16   ,  527.2481 ,  933.5462 , 1129.8474 , 1291.7384 ,
            1258.145  , 1389.6775 , 1327.6411 , 1220.6648 ,  833.171  ,
            1474.9319 , 1452.7163 , 1267.1683 , 1279.886  ,  960.3336 ,
            1276.5159 , 1400.765  , 1677.8501 , 2042.5717 , 2124.2522 ,
            1355.1969 , 1442.7975 , 1082.1064 ,  225.3959 ,  563.7358 ,
             979.7941 , 1151.4218 , 1192.9713 , 1384.5361 ,  850.7755 ,
             447.9525 ,  399.26468, 1159.328  , 1456.6412 , 1527.303  ,
            1444.7692 ,  975.9609 ,  676.3401 ,  786.4795 ,  867.5683 ,
            1708.3972 , 1378.4937 , 1139.245  , 1168.4834 , 1069.594  ,
            1168.8553 , 1902.7328 , 1818.7461 , 1453.2756 , 1852.5608 ,
             887.9272 , 1613.1842 , 1769.9429 , 2205.197  , 1849.0339 ,
            1040.8864 ,  820.7377 ,  329.20322,  929.2557 ,  728.04736,
             625.78015,  210.93626,  451.8591 ,  588.43134,  630.23016,
             667.7374 ,  586.4727 ,  994.94977, 1159.0737 , 1611.9614 ,
            1624.7314 , 1463.0077 , 1841.2853 , 1467.7334 , 1691.9176 ,
            1379.842  ,  912.45123,  979.243  ,  902.42194,  427.20914,
            1156.4613 , 2517.5845 , 2316.0183 , 1618.3004 , 1089.1761 ,
             560.98816,  610.06555,  523.3047 ,  569.11914, 1200.8712 ,
             883.9633 ,  440.76474,  374.67267,  368.11172,  997.8416 ,
            1505.2052 , 1274.928  ,  998.32294,  634.2442 ,  508.078  ,
             848.0488 , 1131.8169 , 1070.3165 , 1601.2533 , 1446.3961 ,
            1237.0332 ,  910.8438 , 2074.2227 , 2232.8381 , 2745.0566 ,
            2651.1729 , 2061.5405 , 1410.6324 ,  549.7682 ,  863.762  ,
             890.6775 , 1493.4208 , 1273.1127 , 1668.0272 ,  876.80225,
             854.4941 ,  746.35114,  929.8469 , 1248.6482 , 1050.4738 ,
            1099.1373 ,  900.2979 ,  765.0181 ,  831.8344 , 1025.2861 ,
            1224.8625 , 1795.1866 , 1430.22   ,  967.6676 ,  815.3979 ,
             726.5763 ,  468.08472, 1141.3434 , 1105.9617 ,  922.31915],
           dtype=float32)
```

```
print(len(yhat_array_unscaled))
```

```
175
```

```
plt.plot(actual[5:],label="Measured")
plt.plot(yhat_array_unscaled,label='Forecasted')
plt.legend()
```

```
mape_with_news_daily=MAPE(y_hat_valid_unscaled,actual[5:])
```

```
#yhat_array
#valid=valid_vector.iloc[:-1,-1]
```

```
#valid.shape
```

```
#Since we are using  4 sequential data to predict 5 , we are only predictiing 40 values
```

```
0      1508.352000
1      4224.136000
2      1644.906000
3      2244.414278
4      4521.991200
          ...
175    2027.758000
176    3645.911000
177    1895.926000
```

```
178     377.736000
179    2140.940000
Name: Sales, Length: 180, dtype: float64
```

```
valid_vector2
```

```
0      1508.352000
1      4224.136000
2      1644.906000
3      2244.414278
4      4521.991200
          ...
175    2027.758000
176    3645.911000
177    1895.926000
178     377.736000
179    2140.940000
Name: Sales, Length: 180, dtype: float64
```

```
print(len(yhat_array_unscaled))
```

```
175
```

```
test_evaluation(yhat_array_unscaled,actual[5:])
```

```
467.6889909056557
```

```
array([1.50835200e+03, 4.22413600e+03, 1.64490600e+03, 2.24441428e+03,
       4.52199120e+03, 3.37400000e+01, 9.04354000e+02, 7.78236000e+02,
       1.14420000e+02, 3.18336980e+03, 3.13500000e+01, 3.80378000e+02,
       1.42402600e+03, 2.24441428e+03, 1.49965200e+03, 4.28875000e+03,
       1.06800000e+01, 3.81600000e+00, 5.04542000e+02, 3.59893400e+03,
       2.13222900e+03, 3.16636000e+02, 2.19187300e+03, 2.10679400e+03,
       2.24441428e+03, 2.28320800e+03, 3.68594400e+03, 5.56314000e+02,
       8.98188000e+02, 2.39996000e+03, 7.98742000e+02, 2.01230200e+03,
       1.14268000e+03, 1.17056400e+03, 1.39129400e+03, 5.23376000e+02,
       3.39559000e+03, 2.24441428e+03, 1.66590000e+02, 1.97129050e+03,
       2.54946800e+03, 2.24441428e+03, 1.62025000e+03, 3.47962400e+03,
       3.84856500e+03, 2.74921000e+03, 2.41337800e+03, 2.91651400e+03,
       3.69308400e+03, 2.64316400e+03, 4.40736000e+02, 1.94987200e+03,
       8.05640000e+01, 9.51728800e+03, 7.07848400e+03, 5.12446000e+02,
       6.92950000e+02, 8.55105400e+03, 5.24540000e+02, 4.59034400e+03,
       6.85056000e+02, 3.61988000e+02, 1.90660000e+02, 6.19053800e+03,
       1.44363200e+03, 2.35468000e+02, 2.24441428e+03, 8.92980000e+02,
       2.24441428e+03, 8.43100000e+02, 6.40193000e+03, 3.65855400e+03,
       2.22703800e+03, 1.98026400e+03, 2.24441428e+03, 2.18432700e+03,
       6.08356000e+02, 3.84100000e+02, 7.64304100e+03, 4.91550000e+02,
```

```
               1.59200000e+01, 4.36734700e+03, 7.28502600e+03, 8.49650000e+02,
               4.97922600e+03, 1.51193000e+03, 1.64818800e+03, 7.35991800e+03,
               3.38172000e+02, 7.87121300e+03, 2.39578600e+03, 6.45046200e+03,
               1.41221300e+03, 1.48657600e+03, 2.24441428e+03, 5.59271000e+02,
               1.94408000e+03, 2.74491000e+02, 7.73764000e+02, 7.10804000e+02,
               1.62671000e+03, 1.07822200e+03, 9.39133000e+02, 1.67121400e+03,
               2.24441428e+03, 4.35606100e+03, 1.49658900e+03, 4.75149200e+03,
               5.47039000e+03, 5.63535400e+03, 8.40580200e+03, 1.34332000e+02,
               1.01794000e+03, 3.47359700e+03, 1.26352000e+02, 2.24441428e+03,
               2.78187020e+03, 1.33385800e+03, 4.53720100e+03, 1.51588770e+04,
               3.35239400e+03, 5.29085000e+02, 2.24441428e+03, 9.99868000e+02,
               1.08632000e+03, 4.08726000e+02, 4.69600000e+01, 4.02573000e+03,
               2.24441428e+03, 2.27103000e+02, 1.25295200e+03, 7.55529000e+02,
               2.24441428e+03, 2.51334300e+03, 3.59214000e+02, 1.82307000e+03,
               2.50664600e+03, 2.39358000e+02, 4.00754800e+03, 3.87355900e+03,
               2.91138600e+03, 6.63342020e+03, 8.34658000e+02, 5.59200000e+02,
               4.75523400e+03, 1.36948828e+04, 1.46975600e+03, 7.39727200e+03,
               2.98827400e+03, 2.23618400e+03, 3.57120000e+01, 1.15310900e+03,
               4.95964100e+03, 3.66615700e+03, 5.04817200e+03, 1.61825400e+03,
               6.91294400e+03, 4.91888000e+02, 2.24441428e+03, 1.24151600e+03,
               1.97370800e+03, 3.19806000e+02, 4.49469000e+02, 1.58089400e+03,
               1.72889200e+03, 7.08726000e+02, 5.56400600e+03, 2.24441428e+03,
               1.81521800e+03, 2.82396500e+03, 2.24441428e+03, 5.80936000e+02,
               3.89771400e+03, 3.06888000e+02, 8.58702000e+02, 2.02775800e+03,
               3.64591100e+03, 1.89592600e+03, 3.77736000e+02, 2.14094000e+03])


def rnn_training_model2(df,df_valid,k):
    x_train=df.values
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(x_train)
    SEQ_LEN = k
    DATA_LEN = scaled.shape[0]# Length
    X_train = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,1)#preparing training data from 0:-6
    for i in range(1,SEQ_LEN):
         X_train = np.append(X_train, scaled[i:-SEQ_LEN+i-1,:].reshape(-1,1,1), axis=1)## Eac
    Y_train = scaled[SEQ_LEN:-1,-1]# We cannot predict 1st 4 Observations as they are used a
    model = Sequential()
    model.add(LSTM(100, input_shape=(X_train.shape[1], X_train.shape[2])))
    model.add(Dense(1))
    model.compile(loss='mse', optimizer='adam',metrics = ['mae'])
    epochs = 100
    validation_split = 0.1
    history = model.fit(X_train, Y_train, batch_size=128,epochs=epochs,validation_split=valid
    x_test=df_valid.values
    x_test2=x_test.reshape(-1,1)
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_valid = scaler.fit_transform(x_test2)
    X_test=scaled_valid[0:-SEQ_LEN-1,:].reshape(-1,1,1)
    for i in range(1,SEQ_LEN):
        X_test = np.append(X_test, scaled_valid[i:-SEQ_LEN+i-1,:].reshape(-1,1,1), axis=1)
    y_hat_valid=model.predict(X_test)
    y_hat_valid_unscaled=scaler.inverse_transform(y_hat_valid)
```

```
    yactual=x_test[SEQ_LEN:-1,]
    mape1=MAPE(y_hat_valid,yactual)
    return mape1

y=rnn_training_model(training_data_us_all,valid_vector_uni,3)
```

```
 Train on 1140 samples, validate on 127 samples
 Epoch 1/100
  768/1140 [===================>.........] - ETA: 0s - loss: 0.0069 - mean_absolute_
   updates = self.state_updates
 1140/1140 [==============================] - 1s 959us/sample - loss: 0.0064 - mean_a
 Epoch 2/100
 1140/1140 [==============================] - 0s 55us/sample - loss: 0.0059 - mean_ab
 Epoch 3/100
 1140/1140 [==============================] - 0s 61us/sample - loss: 0.0058 - mean_ab
 Epoch 4/100
 1140/1140 [==============================] - 0s 53us/sample - loss: 0.0058 - mean_ab
 Epoch 5/100
 1140/1140 [==============================] - 0s 68us/sample - loss: 0.0058 - mean_ab
 Epoch 6/100
 1140/1140 [==============================] - 0s 54us/sample - loss: 0.0058 - mean_ab
 Epoch 7/100
 1140/1140 [==============================] - 0s 51us/sample - loss: 0.0058 - mean_ab
 Epoch 8/100
 1140/1140 [==============================] - 0s 51us/sample - loss: 0.0058 - mean_ab
 Epoch 9/100
 1140/1140 [==============================] - 0s 65us/sample - loss: 0.0058 - mean_ab
 Epoch 10/100
 1140/1140 [==============================] - 0s 50us/sample - loss: 0.0058 - mean_ab
 Epoch 11/100
 1140/1140 [==============================] - 0s 57us/sample - loss: 0.0058 - mean_ab
 Epoch 12/100
 1140/1140 [==============================] - 0s 72us/sample - loss: 0.0058 - mean_ab
 Epoch 13/100
 1140/1140 [==============================] - 0s 53us/sample - loss: 0.0058 - mean_ab
 Epoch 14/100
 1140/1140 [==============================] - 0s 57us/sample - loss: 0.0058 - mean_ab
 Epoch 15/100
 1140/1140 [==============================] - 0s 52us/sample - loss: 0.0058 - mean_ab
 Epoch 16/100
 1140/1140 [==============================] - 0s 51us/sample - loss: 0.0058 - mean_ab
 Epoch 17/100
 1140/1140 [==============================] - 0s 55us/sample - loss: 0.0058 - mean_ab
 Epoch 18/100
 1140/1140 [==============================] - 0s 55us/sample - loss: 0.0058 - mean_ab
 Epoch 19/100
 1140/1140 [==============================] - 0s 54us/sample - loss: 0.0058 - mean_ab
 Epoch 20/100
 1140/1140 [==============================] - 0s 68us/sample - loss: 0.0058 - mean_ab
 Epoch 21/100
 1140/1140 [==============================] - 0s 56us/sample - loss: 0.0058 - mean_ab
 Epoch 22/100
 1140/1140 [==============================] - 0s 53us/sample - loss: 0.0058 - mean_ab
 Epoch 23/100
 1140/1140 [==============================] - 0s 70us/sample - loss: 0.0058 - mean_ab
```

```
Epoch 24/100
1140/1140 [==============================] - 0s 54us/sample - loss: 0.0058 - mean_ab
Epoch 25/100
1140/1140 [==============================] - 0s 59us/sample - loss: 0.0058 - mean_ab
Epoch 26/100
1140/1140 [==============================] - 0s 52us/sample - loss: 0.0058 - mean_ab
Epoch 27/100
```

```
y2=rnn_training_model2(training_data_us_all,valid_vector_uni,3)
```

```
Train on 1140 samples, validate on 127 samples
Epoch 1/100
1140/1140 [==============================] - ETA: 0s - loss: 0.0066 - mean_absolute_
  updates = self.state_updates
1140/1140 [==============================] - 3s 3ms/sample - loss: 0.0066 - mean_abs
Epoch 2/100
1140/1140 [==============================] - 0s 240us/sample - loss: 0.0060 - mean_a
Epoch 3/100
1140/1140 [==============================] - 0s 226us/sample - loss: 0.0058 - mean_a
Epoch 4/100
1140/1140 [==============================] - 0s 245us/sample - loss: 0.0058 - mean_a
Epoch 5/100
1140/1140 [==============================] - 0s 200us/sample - loss: 0.0058 - mean_a
Epoch 6/100
1140/1140 [==============================] - 0s 215us/sample - loss: 0.0058 - mean_a
Epoch 7/100
1140/1140 [==============================] - 0s 289us/sample - loss: 0.0058 - mean_a
Epoch 8/100
1140/1140 [==============================] - 0s 220us/sample - loss: 0.0058 - mean_a
Epoch 9/100
1140/1140 [==============================] - 0s 255us/sample - loss: 0.0058 - mean_a
Epoch 10/100
1140/1140 [==============================] - 0s 230us/sample - loss: 0.0058 - mean_a
Epoch 11/100
1140/1140 [==============================] - 0s 191us/sample - loss: 0.0058 - mean_a
Epoch 12/100
1140/1140 [==============================] - 0s 186us/sample - loss: 0.0058 - mean_a
Epoch 13/100
1140/1140 [==============================] - 0s 287us/sample - loss: 0.0058 - mean_a
Epoch 14/100
1140/1140 [==============================] - 0s 305us/sample - loss: 0.0058 - mean_a
Epoch 15/100
1140/1140 [==============================] - 0s 322us/sample - loss: 0.0058 - mean_a
Epoch 16/100
1140/1140 [==============================] - 0s 315us/sample - loss: 0.0058 - mean_a
Epoch 17/100
1140/1140 [==============================] - 0s 295us/sample - loss: 0.0058 - mean_a
Epoch 18/100
1140/1140 [==============================] - 0s 304us/sample - loss: 0.0058 - mean_a
Epoch 19/100
1140/1140 [==============================] - 0s 225us/sample - loss: 0.0058 - mean_a
Epoch 20/100
1140/1140 [==============================] - 0s 186us/sample - loss: 0.0058 - mean_a
Epoch 21/100
```

```
1140/1140 [==============================] - 0s 168us/sample - loss: 0.0058 - mean_a
Epoch 22/100
1140/1140 [==============================] - 0s 184us/sample - loss: 0.0058 - mean_a
Epoch 23/100
1140/1140 [==============================] - 0s 206us/sample - loss: 0.0058 - mean_a
Epoch 24/100
1140/1140 [==============================] - 0s 212us/sample - loss: 0.0058 - mean_a
Epoch 25/100
1140/1140 [==============================] - 0s 138us/sample - loss: 0.0058 - mean_a
Epoch 26/100
1140/1140 [==============================] - 0s 197us/sample - loss: 0.0058 - mean_a
Epoch 27/100
```

```
y2
```

```
99.97129767047426
```

```
List=['RNN-Uni,50 Dense','RNN-50-Multiv','RNN Uni-100 dense ']
mape_method=[y,mape_with_news_daily,y2]
plt.bar(List,mape_method)
plt.xlabel("Type of Forecast")
plt.ylabel("MAPE_Loss Function")
plt.title("Impact of News on Forecasting")
```

```
x_train=training_data_us_all.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(x_train)
SEQ_LEN = 5
DATA_LEN = scaled.shape[0]# Length
X_train = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,1)#preparing training data from 0:-6
for i in range(1,SEQ_LEN):
        X_train = np.append(X_train, scaled[i:-SEQ_LEN+i-1,:].reshape(-1,1,1), axis=1)## Eac
Y_train = scaled[SEQ_LEN:-1,-1]# We cannot predict 1st 4 Observations as they are used a feat
```

```
model = Sequential()
model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam',metrics = ['mae'])
epochs =500
validation_split = 0.1
history = model.fit(X_train, Y_train, batch_size=128,epochs=epochs,validation_split=validatio
```

```
Train on 1138 samples, validate on 127 samples
Epoch 1/500
1138/1138 [==============================] - 1s 544us/sample - loss: 0.0066 - mean_a
/usr/local/lib/python3.7/dist-packages/keras/engine/training_v1.py:2057: UserWarning
  updates = self.state_updates
Epoch 2/500
1138/1138 [==============================] - 0s 96us/sample - loss: 0.0060 - mean_ab
Epoch 3/500
1138/1138 [==============================] - 0s 78us/sample - loss: 0.0058 - mean_ab
Epoch 4/500
1138/1138 [==============================] - 0s 70us/sample - loss: 0.0058 - mean_ab
Epoch 5/500
1138/1138 [==============================] - 0s 80us/sample - loss: 0.0058 - mean_ab
Epoch 6/500
1138/1138 [==============================] - 0s 74us/sample - loss: 0.0058 - mean_ab
Epoch 7/500
1138/1138 [==============================] - 0s 72us/sample - loss: 0.0058 - mean_ab
Epoch 8/500
1138/1138 [==============================] - 0s 79us/sample - loss: 0.0058 - mean_ab
Epoch 9/500
1138/1138 [==============================] - 0s 74us/sample - loss: 0.0058 - mean_ab
Epoch 10/500
1138/1138 [==============================] - 0s 74us/sample - loss: 0.0058 - mean_ab
Epoch 11/500
1138/1138 [==============================] - 0s 82us/sample - loss: 0.0058 - mean_ab
Epoch 12/500
1138/1138 [==============================] - 0s 72us/sample - loss: 0.0058 - mean_ab
Epoch 13/500
1138/1138 [==============================] - 0s 84us/sample - loss: 0.0058 - mean_ab
Epoch 14/500
1138/1138 [==============================] - 0s 73us/sample - loss: 0.0058 - mean_ab
Epoch 15/500
1138/1138 [==============================] - 0s 74us/sample - loss: 0.0058 - mean_ab
Epoch 16/500
1138/1138 [==============================] - 0s 80us/sample - loss: 0.0058 - mean_ab
Epoch 17/500
1138/1138 [==============================] - 0s 74us/sample - loss: 0.0058 - mean_ab
Epoch 18/500
1138/1138 [==============================] - 0s 74us/sample - loss: 0.0058 - mean_ab
Epoch 19/500
1138/1138 [==============================] - 0s 74us/sample - loss: 0.0058 - mean_ab
Epoch 20/500
1138/1138 [==============================] - 0s 77us/sample - loss: 0.0058 - mean_ab
Epoch 21/500
1138/1138 [==============================] - 0s 72us/sample - loss: 0.0058 - mean_ab
Epoch 22/500
1138/1138 [==============================] - 0s 75us/sample - loss: 0.0058 - mean_ab
```

```
Epoch 23/500
1138/1138 [==============================] - 0s 73us/sample - loss: 0.0058 - mean_ab
Epoch 24/500
1138/1138 [==============================] - 0s 80us/sample - loss: 0.0058 - mean_ab
Epoch 25/500
1138/1138 [==============================] - 0s 81us/sample - loss: 0.0058 - mean_ab
Epoch 26/500
1138/1138 [==============================] - 0s 84us/sample - loss: 0.0058 - mean_ab
Epoch 27/500
```

```
plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'], label='training')
plt.plot(history.epoch,history.history['val_loss'], label='validation')
plt.title('loss')
plt.legend(loc='best')
```

```
x_test=valid_vector_uni.values
x_test2=x_test.reshape(-1,1)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_valid = scaler.fit_transform(x_test2)
X_test=scaled_valid[0:-SEQ_LEN-1,:].reshape(-1,1,1)
for i in range(1,SEQ_LEN):
        X_test = np.append(X_test, scaled_valid[i:-SEQ_LEN+i-1,:].reshape(-1,1,1), axis=1)
y_hat_valid=model.predict(X_test)
y_hat_valid_unscaled=scaler.inverse_transform(y_hat_valid)
#yactual=x_test[SEQ_LEN:-1,-1]
#mape1=MAPE(y_hat_valid,yactual)


y_actual=x_test[SEQ_LEN:-1]


plt.plot(actual[5:],label="Measured")
plt.plot(yhat_array_unscaled,label='Forecasted')
plt.legend()
```