```
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
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages (0
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: scipy>=1.3 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (
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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
```

```
!apt-get install rar

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

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
rar is already the newest version (2:5.5.0-1).
The following packages were automatically installed and are no longer required:
  libnvidia-common-460 nsight-compute-2020.2.0
Use 'apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 42 not upgraded.
```

```
RAR 5.50   Copyright (c) 1993-2017 Alexander Roshal   11 Aug 2017
Trial version             Type 'rar -?' for help

Evaluation copy. Please register.

ERROR: Bad archive /content/drive/My Drive/Wisconsin_Project/superstore_dataset2011-2015

Program aborted
```

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

```
Archive:  /content/drive/My Drive/Wisconsin_Project/superstore_dataset2011-2015.csv.zip
replace superstore_dataset2011-2015.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
```

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

```
Archive:  /content/drive/My Drive/Wisconsin_Project/Combined_News_DJIA.csv.zip
replace Combined_News_DJIA.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: n
```

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

```
external_news=pd.read_csv('Combined_News_DJIA.csv')
```

```
external_news.shape
```

```
(1989, 27)
```

```
external_news.tail(10)
```

```
external_news.head(10)
```

```
data.head(2)
```

```
data.dtypes
```

```
Row ID              int64
Order ID           object
Order Date         object
Ship Date          object
Ship Mode          object
Customer ID        object
Customer Name      object
Segment            object
City               object
State              object
Country            object
Postal Code       float64
Market             object
Region             object
Product ID         object
Category           object
Sub-Category       object
Product Name       object
Sales             float64
Quantity            int64
Discount          float64
Profit            float64
Shipping Cost     float64
Order Priority     object
dtype: object
```

```
data['Market'].value_counts()
```

```
APAC      11002
LATAM     10294
EU        10000
US         9994
```

```
EMEA       5029
Africa     4587
Canada      384
Name: Market, dtype: int64
```

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

```
dtype('O')
```

```
data.iloc[0:20067,2]
```

```
0          1/1/2011
1          1/1/2011
2          1/1/2011
3          1/1/2011
4          1/1/2011
              ...
20062    12/12/2014
20063    12/12/2014
20064    12/12/2014
20065    12/12/2014
20066    12/12/2014
Name: Order Date, Length: 20067, dtype: object
```

```
#standardization for 1st part of data
data.iloc[0:20067,2]=pd.to_datetime(data.iloc[0:20067,2], format='%m/%d/%Y')#Y REPRESRENTS YE
```

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

```
# Remove Time
data['Order Date'] = pd.to_datetime(data['Order Date']).dt.date
```

```
data
```

```
data["Order_Date_c"] = pd.to_datetime(data["Order Date"], utc=True)# convert into date to dat
```

```
data['Year']=data['Order_Date_c'].dt.strftime("%Y")# Only Order Year
data['Month']=data['Order_Date_c'].dt.strftime("%m")
data['Day']=data['Order_Date_c'].dt.strftime("%d")
```

```
data.head(3)
```

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

```
persistency_date=pd.to_datetime('2014-01-01').date()
persistency_data=data.loc[(data['Order Date']> persistency_date) & (data['Order Date'] < vali
```

```
persistency_cohort=['Market','Category']
persistency_data_check=persistency_data.groupby(persistency_cohort).agg(History_Sales=('Sales
```

```
validation_data.tail(1)
```

EXTERNAL NEWS DATA SET PREPERATION

```
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.head(2)
```

```
validation_data.index
```

```
Int64Index([16589, 16590, 16591, 16592, 16593, 16594, 16595, 16596, 16597,
            16598,
            ...
            51280, 51281, 51282, 51283, 51284, 51285, 51286, 51287, 51288,
            51289],
           dtype='int64', length=10186)
```

```
validation_data.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', 'Order_Date_c', 'Year',
       'Month', 'Day'],
      dtype='object')
```

```
training_data = data.drop(labels=validation_data.index, axis=0)
```

```
training_data.head(3)
```

```
training_data
```

```
training_data.index
```

```
Int64Index([    0,     1,     2,     3,     4,     5,     6,     7,     8,
                9,
            ...
            51218, 51219, 51220, 51221, 51222, 51223, 51224, 51225, 51226,
            51227],
           dtype='int64', length=41104)
```

```
training_data.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', 'Order_Date_c', 'Year',
       'Month', 'Day'],
      dtype='object')
```

```
consumer_vector=['Region',"Market","Country","Category","Year","Month"]
```

```
consumer_vector2=['Region',"Market","Country","Category"]
```

```
df_cv=training_data.groupby(consumer_vector2)['Order Date'].count().reset_index().rename(colu
```

```
df_cv
```

```
sns.histplot(df_cv['Series_Length'])
```

```
Vector2=["Market","Category"]
```

```
valid_data=validation_data.groupby(consumer_vector).agg(Monthly_Quantity=('Sales','sum')).res
```

```
df1=training_data.groupby(Vector2)['Order Date'].count().reset_index().rename(columns={'Order
```

```
df2=training_data.groupby(Vector2)["Sales"].apply(lambda column: ((column == 0)/column.count(
```

```
df3=training_data.groupby(Vector2)["Sales"].apply(lambda x: np.std(x, ddof=1) / (np.mean(x))
```

```
df_temp =[df.set_index(Vector2) for df in [df1,df2,df3]]
df_temp =[df.set_index(Vector2) for df in [df1,df2,df3]]
df_fet = pd.concat(df_temp, axis=1).reset_index()
```

```
df_fet
```

```
sns.histplot(df_fet['Series_Length'])
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

```
df_fet["Market"] = df_fet["Market"].astype('category')
df_fet["Market"] =df_fet["Market"].cat.codes
df_fet["Category"]=df_fet["Category"].astype('category')
df_fet["Category"]=df_fet["Category"].cat.codes
```

```
df2=df_fet.iloc[:,[0,1,2,4,5]]
```

```
df2
```

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

X_train=df2.iloc[:,0:]
mse = []
#Fitting on Training data
for i in range(1, 12):
    kmeans = KMeans(n_clusters=i,  random_state=0,init='k-means++')
    kmeans.fit(X_train)
    mse.append(kmeans.inertia_)
fig = plt.figure(figsize=(5,5))
plt.plot(range(1, 12), mse)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('mse')
plt.show()
```

```
kmeans = KMeans(n_clusters=2).fit(X_train)
centroids = kmeans.cluster_centers_
print(centroids)

plt.scatter(X_train['Series_Length'], X_train['Variability'], c= kmeans.labels_.astype(float)
plt.scatter(centroids[:, 0], centroids[:, 1], c='red')
plt.show()
```

```
kmeans.fit(X_train)
predict=kmeans.predict(X_train)
```

```
predict
X_train['Cluster_Flag'] = pd.Series(predict, index=X_train.index)
```

```
X_train
```

```
training_data.index= pd.DatetimeIndex(training_data['Order Date'])
training_data2=training_data.drop('Order Date',axis=1)
```

```
validation_data.index=pd.DatetimeIndex(validation_data['Order Date'])
validation_data2=validation_data.drop('Order Date',axis=1)
```

```
training_data2.groupby(Vector2).count().reset_index().rename(columns={'Sales':'Series_Length'
```

```
Market_List=training_data2['Market'].unique()
Market_List
```

```
array(['Africa', 'APAC', 'EMEA', 'EU', 'US', 'LATAM', 'Canada'],
      dtype=object)
```

```
cohort_1=training_data2[(training_data2['Market']=='US') & (training_data2['Category']=='Offi
```

```
cohort_2=training_data2[(training_data2['Market']=='US') & (training_data2['Category']=='Tech
```

```
cohort_3=training_data2[(training_data2['Market']=='US') & (training_data2['Category']=='Furn
```

```
cohort_4=training_data2[(training_data2['Market']=='EMEA') & (training_data2['Category']=='Te
```

```
monthly_cohort_1=cohort_1.resample('m').sum()
```

```
monthly_cohort_2=cohort_2.resample('m').sum()
```

```
monthly_cohort_3=cohort_3.resample('m').sum()
```

```
weekly_cohort=cohort_1.resample('W').mean()
```

```
weekly_cohort2=cohort_2.resample('W').mean()
```

```
x_week=weekly_cohort['Sales']
```

```
type(training_data2.index)
```

```
pandas.core.indexes.datetimes.DatetimeIndex
```

```
x_m_1=monthly_cohort_1['Sales']
x_m_2=monthly_cohort_2['Sales']
x_m_3=monthly_cohort_2['Sales']
```

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(x_m_1, lags=30, zero=False, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(x_m_1, lags=30, zero=False, ax=ax2)
```

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(x_m_2, lags=30, zero=False, ax=ax1)
ax2 = fig.add_subplot(212)
ax2.set_title("Technology")
fig = sm.graphics.tsa.plot_pacf(x_m_2, lags=30, zero=False, ax=ax2)
```

```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(x_m_3, lags=30, zero=False, ax=ax1)
ax2 = fig.add_subplot(212)
ax2.set_title("Technology")
fig = sm.graphics.tsa.plot_pacf(x_m_3, lags=30, zero=False, ax=ax2)
```

```
x_m_3,
```

```
(Order Date
 2011-01-31     7931.580
 2011-02-28     6484.728
 2011-03-31    33152.762
 2011-04-30     9425.530
 2011-05-31     7595.200
```

```
2011-06-30     6741.179
2011-07-31     8539.334
2011-08-31    16900.912
2011-09-30    24338.380
2011-10-31    12865.386
2011-11-30    20742.904
2011-12-31    20560.338
2012-01-31     6974.476
2012-02-29     6845.640
2012-03-31     9791.866
2012-04-30    15310.406
2012-05-31     7328.590
2012-06-30     9581.120
2012-07-31    10024.190
2012-08-31    24198.782
2012-09-30    22681.540
2012-10-31     9166.250
2012-11-30    11836.373
2012-12-31    29041.576
2013-01-31     4212.320
2013-02-28    19287.712
2013-03-31    28439.960
2013-04-30    16688.356
2013-05-31    34929.566
2013-06-30    12211.458
2013-07-31    15602.789
2013-08-31    19152.620
2013-09-30    12449.212
2013-10-31     9235.440
2013-11-30    33057.445
2013-12-31    20794.926
2014-01-31    22858.943
2014-02-28    15121.738
2014-03-31    38463.480
2014-04-30    16162.071
2014-05-31    19964.464
2014-06-30    16585.262
Freq: M, Name: Sales, dtype: float64,)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(x_week, lags=100, zero=False, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(x_week, lags=49, zero=False, ax=ax2)
```

AUTO CORRELATION PLOT OF WEEKLY DATA

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(x_week, lags=100, zero=False, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(x_week, lags=49, zero=False, ax=ax2)
```

```
consumer_vector=['Region','Market','Category','Country','Year','Month']
```

```
training_data
```

```
from numpy.core.fromnumeric import mean
data3=training_data.groupby(consumer_vector).agg(Monthly_Quantity=('Sales','sum')).reset_inde
```

```
data3['Region'].unique()
```

```
array(['Africa', 'Canada', 'Caribbean', 'Central', 'Central Asia', 'EMEA',
       'East', 'North', 'North Asia', 'Oceania', 'South',
       'Southeast Asia', 'West'], dtype=object)
```

```
data3
```

```
data3
```

```
data3['Category'].value_counts()
```

```
Office Supplies    3045
Technology         2162
Furniture          2066
Name: Category, dtype: int64
```

```
data4=data3[(data3['Region']=='West') & (data3['Category']=='Furniture')]
```

```
pd.to_datetime(data4.loc[:,['Year','Month']].assign(DAY=1))
```

```
7147   2011-01-01
7148   2011-02-01
7149   2011-03-01
7150   2011-04-01
7151   2011-05-01
7152   2011-06-01
7153   2011-07-01
7154   2011-08-01
7155   2011-09-01
7156   2011-10-01
7157   2011-11-01
7158   2011-12-01
7159   2012-01-01
7160   2012-02-01
7161   2012-03-01
7162   2012-04-01
7163   2012-05-01
7164   2012-06-01
7165   2012-07-01
7166   2012-08-01
7167   2012-09-01
7168   2012-10-01
7169   2012-11-01
7170   2012-12-01
7171   2013-01-01
7172   2013-02-01
7173   2013-03-01
```

```
7174   2013-04-01
7175   2013-05-01
7176   2013-06-01
7177   2013-07-01
7178   2013-08-01
7179   2013-09-01
7180   2013-10-01
7181   2013-11-01
7182   2013-12-01
7183   2014-01-01
7184   2014-02-01
7185   2014-03-01
7186   2014-04-01
7187   2014-05-01
7188   2014-06-01
dtype: datetime64[ns]
```

```
data4=data3[(data3['Region']=='West') & (data3['Category']=='Furniture')]
data4['Synthetic_Date'] = pd.to_datetime(data4.loc[:,['Year','Month']].assign(DAY=1))
data4.shape
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: 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_

(42, 8)
```

```
data4
```

```
#data4=data3[(data3['Region']=='West') & (data3['Category']=='Furniture')]
```

```
data4
```

```
data4_gp2=data3[(data3['Market']=='US') & (data3['Category']=='Technology')]
#data4_gp2['Synthetic_Date'] =  pd.to_datetime(data4.loc[:,['Year','Month']].assign(DAY=1))
```

```
data4_gp2
pd.to_datetime(data4_gp2.loc[:,['Year','Month']].assign(DAY=1))
data4_gp2['Synthetic_Date']=pd.to_datetime(data4_gp2.loc[:,['Year','Month']].assign(DAY=1))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: 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_
  This is separate from the ipykernel package so we can avoid doing imports until
```

```
data4_gp2['Synthetic_Date']
```

```
2885   2011-02-01
2886   2011-03-01
2887   2011-04-01
2888   2011-05-01
2889   2011-06-01
          ...
7268   2014-02-01
7269   2014-03-01
7270   2014-04-01
7271   2014-05-01
7272   2014-06-01
Name: Synthetic_Date, Length: 166, dtype: datetime64[ns]
```

```
data4_gp2
```

```
data4_gp2
```

```
data4_gp3=data3[(data3['Country']=='United States') & (data3['Category']=='Office Supplies')]
data4_gp3['Synthetic_Date'] = pd.to_datetime(data4_gp3[['Year', 'Month']].assign(DAY=1))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: 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_
```

```
data4_gp2
```

```
def time_index(df):
  df_n=df.iloc[:,6:8]
  q=df_n.set_index('Synthetic_Date')
  return q
```

Three data sets for 3 systems

DF_FIRST REPRESENTS US FURNITURES DF_SECOND REPRESENTS US TECHNOLOGY DF_THIRD REPRESENTS US OFFICE SUPPLIES

```
df_second=time_index(data4_gp2)
df_first=time_index(data4)
df_third=time_index(data4_gp3)
```

```
def time_object(df):
  df_n=df.iloc[:,6:8]
  return df_n
```

```
df1=time_object(data4)#Furniture
df2=time_object(data4_gp2)#technology
df3=time_object(data4_gp3)#Office Supplies
```

```
s=df1.set_index('Synthetic_Date')['Monthly_Quantity']
```

```
s2_tech=df2.set_index('Synthetic_Date')['Monthly_Quantity']
s3_office=df3.set_index('Synthetic_Date')['Monthly_Quantity']
```

```
df_first
```

```
#df_first.index=df_first['Synthetic_Date']

#df_second.index=df_second['Synthetic_Date']

#df_third.index=df_third['Synthetic_Date']

#del df_second['Synthetic_Date']

#del df_first['Synthetic_Date']


#del df_third['Synthetic_Date']

#df_first

#df_second

#df_third

df_first.dropna()
```

```
s

    Synthetic_Date
    2011-01-01     1674.2030
```

```
2011-02-01      202.8880
2011-03-01     3496.5940
2011-04-01     2965.6380
2011-05-01     1955.6020
2011-06-01     2032.8220
2011-07-01     5836.9660
2011-08-01     3830.8080
2011-09-01     6603.5670
2011-10-01     4470.3000
2011-11-01     5102.0585
2011-12-01    11911.0015
2012-01-01    11172.6980
2012-02-01     2944.6570
2012-03-01     6845.6910
2012-04-01     1115.4960
2012-05-01     4954.8785
2012-06-01     2580.6740
2012-07-01     5015.9110
2012-08-01     3529.0110
2012-09-01     5371.7500
2012-10-01     4263.8270
2012-11-01     5262.5460
2012-12-01     3947.8040
2013-01-01     4199.6300
2013-02-01      243.5480
2013-03-01     3813.1820
2013-04-01     8063.8470
2013-05-01     5738.4710
2013-06-01     6359.0470
2013-07-01     2624.8220
2013-08-01     7763.1480
2013-09-01     9438.6185
2013-10-01     2928.6430
2013-11-01     9244.8700
2013-12-01    13401.8140
2014-01-01     1804.4130
2014-02-01     6922.5420
2014-03-01     5467.3000
2014-04-01     6588.4280
2014-05-01     5135.3720
2014-06-01     3631.9005
Name: Monthly_Quantity, dtype: float64
```

```
from statsmodels.tsa.seasonal import seasonal_decompose
series = s
result = seasonal_decompose(series, model='additive')
result.plot()
```

```
df_first
```

```
df_first['2013']
```

```
df_first.plot()
# We see we dont have much data to look into seasoanlity
```

```
from statsmodels.tsa.stattools import adfuller
test_result=adfuller(df_first['Monthly_Quantity'])
test_result
```

```
(-5.468367670395494,
 2.429887304788359e-06,
 0,
 41,
 {'1%': -3.60098336718852,
  '10%': -2.6059629803688282,
  '5%': -2.9351348158036012},
 589.5166599193723)
```

```
def adfuller_test(sales):
    result=adfuller(sales)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. D
    else:
```

```
        print("weak evidence against null hypothesis,indicating it is non-stationary ")
    return
```

```
adfuller_test(df_first['Monthly_Quantity'])
```

```
ADF Test Statistic : -5.468367670395494
p-value : 2.429887304788359e-06
#Lags Used : 0
Number of Observations : 41
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data is sta
```

```
from statsmodels.tsa.arima_model import ARIMA
from pandas import datetime
from pandas import read_csv
from pandas import DataFrame
from matplotlib import pyplot
# fit model
model = ARIMA(df_first, order=(3,2,0))
model_fit = model.fit()
# summary of fit model
print(model_fit.summary())
# line plot of residuals
residuals = DataFrame(model_fit.resid)
residuals.plot()
pyplot.show()
# density plot of residuals
residuals.plot(kind='kde')
pyplot.show()
# summary stats of residuals
print(residuals.describe())
```

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df_first['Monthly_Quantity'], lags=35, zero=False, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df_first['Monthly_Quantity'], lags=18, zero=False, ax=ax2)
```

```
#There is no correlation as seen above neither between lags- but we can see some seasonality/
```

```
#There is some seasonality
```

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
# Create a SARIMAX model
model = SARIMAX(df_first['Monthly_Quantity'], order=(2, 1, 0), seasonal_order=(1, 1, 0, 7))

# Fit the model
results = model.fit()

# Print the results summary
results.summary()
```

```
pip install pmdarima
```

```
Requirement already satisfied: pmdarima in /usr/local/lib/python3.7/dist-packages (1.8.5
Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: Cython!=0.29.18,>=0.29 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in /usr/local/lib/python3.7/di
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from p
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>=1.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist
```

```
import pmdarima as pmd

def arimamodel(timeseriesarray):
    autoarima_model = pmd.auto_arima(timeseriesarray,
                              start_p=1,
                              start_q=1,max_p=3,max_d=3,max_q=3,
                              test="adf",start_P=0,D=1,start_Q=0,max_P=3,max_D=3,max_Q=3,m=12
                              trace=True,seasonal=True,stepwise=True,suppress_warnings=True)
    return autoarima_model
```

```
import pmdarima as pmd

def arimamodel2(timeseriesarray):
    autoarima_model = pmd.auto_arima(timeseriesarray,
                                start_p=1,
                                start_q=1,max_p=3,max_d=2,max_q=3,
                                test="adf",start_P=0,D=2,start_Q=0,max_P=3,max_D=2,max_Q=3,m=4,
                                trace=True,seasonal=True,stepwise=True,suppress_warnings=True)
    return autoarima_model
```

```
arimamodel(df_first)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,2,1)(0,1,0)[12]             : AIC=inf, Time=0.14 sec
 ARIMA(0,2,0)(0,1,0)[12]             : AIC=591.823, Time=0.01 sec
 ARIMA(1,2,0)(1,1,0)[12]             : AIC=569.514, Time=0.08 sec
 ARIMA(0,2,1)(0,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,2,0)(0,1,0)[12]             : AIC=573.839, Time=0.02 sec
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py:522: ModelFit
Traceback:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py", lin
    fit.fit(y, X=X, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 597, i
    self._fit(y, X, **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 518, i
    fit, self.arima_res_ = _fit_wrapper()
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 512, i
    **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/mlemodel.p
    @loglikelihood_burn.setter
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
    warning_description = ' for %s' % warning_description
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py", line 40
    else:
ValueError: maxlag should be < nobs

  warnings.warn(warning_str, ModelFitWarning)
 ARIMA(1,2,0)(2,1,0)[12]             : AIC=571.172, Time=0.29 sec
 ARIMA(1,2,0)(1,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,2,0)(0,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,2,0)(2,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(0,2,0)(1,1,0)[12]             : AIC=590.385, Time=0.14 sec
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py:522: ModelFit
Traceback:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py", lin
    fit.fit(y, X=X, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 597, i
    self._fit(y, X, **fit_args)
```

```
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 518, i
    fit, self.arima_res_ = _fit_wrapper()
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 512, i
    **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/mlemodel.py
    @loglikelihood_burn.setter
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
    warning_description = ' for %s' % warning_description
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py", line 40
    else:
ValueError: maxlag should be < nobs

  warnings.warn(warning_str, ModelFitWarning)
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py:522: ModelFit
Traceback:
Traceback (most recent call last):
```

```
arimamodel(df_third)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,1,0)[12] intercept   : AIC=2967.189, Time=0.16 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=2971.459, Time=0.01 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=2914.249, Time=0.93 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=1.63 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=2969.741, Time=0.03 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=2965.971, Time=0.06 sec
 ARIMA(1,0,0)(2,1,0)[12] intercept   : AIC=2902.195, Time=2.05 sec
 ARIMA(1,0,0)(3,1,0)[12] intercept   : AIC=2897.397, Time=5.58 sec
 ARIMA(1,0,0)(3,1,1)[12] intercept   : AIC=2899.734, Time=14.44 sec
 ARIMA(1,0,0)(2,1,1)[12] intercept   : AIC=2896.929, Time=7.77 sec
 ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=3.17 sec
 ARIMA(1,0,0)(2,1,2)[12] intercept   : AIC=inf, Time=6.15 sec
 ARIMA(1,0,0)(1,1,2)[12] intercept   : AIC=inf, Time=5.08 sec
 ARIMA(1,0,0)(3,1,2)[12] intercept   : AIC=inf, Time=10.46 sec
 ARIMA(0,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=4.02 sec
 ARIMA(2,0,0)(2,1,1)[12] intercept   : AIC=2897.324, Time=5.94 sec
 ARIMA(1,0,1)(2,1,1)[12] intercept   : AIC=2901.302, Time=5.03 sec
 ARIMA(0,0,1)(2,1,1)[12] intercept   : AIC=inf, Time=4.30 sec
 ARIMA(2,0,1)(2,1,1)[12] intercept   : AIC=2900.133, Time=6.62 sec
 ARIMA(1,0,0)(2,1,1)[12]             : AIC=inf, Time=2.97 sec

Best model:  ARIMA(1,0,0)(2,1,1)[12] intercept
Total fit time: 86.439 seconds
ARIMA(order=(1, 0, 0), scoring_args={}, seasonal_order=(2, 1, 1, 12),
      suppress_warnings=True)
```

```
results_furniture=arimamodel2(df_first)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,2,0)[4]             : AIC=707.248, Time=0.05 sec
 ARIMA(0,0,0)(0,2,0)[4]             : AIC=703.456, Time=0.01 sec
 ARIMA(1,0,0)(1,2,0)[4]             : AIC=inf, Time=0.13 sec
```

```
 ARIMA(0,0,1)(0,2,1)[4]             : AIC=inf, Time=0.21 sec
 ARIMA(0,0,0)(1,2,0)[4]             : AIC=677.836, Time=0.02 sec
 ARIMA(0,0,0)(2,2,0)[4]             : AIC=676.102, Time=0.04 sec
 ARIMA(0,0,0)(3,2,0)[4]             : AIC=675.548, Time=0.11 sec
 ARIMA(0,0,0)(3,2,1)[4]             : AIC=inf, Time=0.57 sec
 ARIMA(0,0,0)(2,2,1)[4]             : AIC=inf, Time=0.17 sec
 ARIMA(1,0,0)(3,2,0)[4]             : AIC=672.912, Time=0.70 sec
 ARIMA(1,0,0)(2,2,0)[4]             : AIC=673.790, Time=0.37 sec
 ARIMA(1,0,0)(3,2,1)[4]             : AIC=inf, Time=0.74 sec
 ARIMA(1,0,0)(2,2,1)[4]             : AIC=inf, Time=0.42 sec
 ARIMA(2,0,0)(3,2,0)[4]             : AIC=674.739, Time=0.79 sec
 ARIMA(1,0,1)(3,2,0)[4]             : AIC=inf, Time=0.79 sec
 ARIMA(0,0,1)(3,2,0)[4]             : AIC=673.063, Time=0.34 sec
 ARIMA(2,0,1)(3,2,0)[4]             : AIC=676.716, Time=0.81 sec
 ARIMA(1,0,0)(3,2,0)[4] intercept   : AIC=inf, Time=nan sec

Best model:  ARIMA(1,0,0)(3,2,0)[4]
Total fit time: 6.757 seconds
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py:522: ModelFit
Traceback:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py", lin
    fit.fit(y, X=X, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 597, i
    self._fit(y, X, **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 518, i
    fit, self.arima_res_ = _fit_wrapper()
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 512, i
    **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/mlemodel.py
    # Initialization (this is done here rather than in the constructor
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py", line 470,
    Line-search error tolerance
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/base/optimizer.py", line
    methods += extra_fit_funcs.keys()
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/base/optimizer.py", line 4
    newparams = oldparams - np.linalg.solve(H, score(oldparams))
  File "/usr/local/lib/python3.7/dist-packages/scipy/optimize/lbfgsb.py", line 199,
    **opts)
  File "/usr/local/lib/python3.7/dist-packages/scipy/optimize/lbfgsb.py", line 345,
    f, g = func_and_grad(x)
  File "/usr/local/lib/python3.7/dist-packages/scipy/optimize/lbfgsb.py", line 290,
    f = fun(x, *args)
  File "/usr/local/lib/python3.7/dist-packages/scipy/optimize/optimize.py", line 327
    return function(*(wrapper_args + args))
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py", line 444,
    (min, max) pairs for each element in x,
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/mlemodel.py
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/kalman_fil
    obs_cov = self['obs_cov']
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/kalman_fil
    all intermediate matrices are stored.
```

```
results_tech=arimamodel2(df_second)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,2,0)[4]             : AIC=inf, Time=0.24 sec
 ARIMA(0,0,0)(0,2,0)[4]             : AIC=3361.093, Time=0.01 sec
 ARIMA(1,0,0)(1,2,0)[4]             : AIC=3282.350, Time=0.19 sec
 ARIMA(0,0,1)(0,2,1)[4]             : AIC=inf, Time=0.49 sec
 ARIMA(1,0,0)(0,2,0)[4]             : AIC=3362.668, Time=0.05 sec
 ARIMA(1,0,0)(2,2,0)[4]             : AIC=3270.226, Time=0.25 sec
 ARIMA(1,0,0)(3,2,0)[4]             : AIC=inf, Time=1.80 sec
 ARIMA(1,0,0)(2,2,1)[4]             : AIC=inf, Time=1.58 sec
 ARIMA(1,0,0)(1,2,1)[4]             : AIC=inf, Time=1.52 sec
 ARIMA(1,0,0)(3,2,1)[4]             : AIC=inf, Time=3.38 sec
 ARIMA(0,0,0)(2,2,0)[4]             : AIC=3215.448, Time=0.24 sec
 ARIMA(0,0,0)(1,2,0)[4]             : AIC=3248.298, Time=0.20 sec
 ARIMA(0,0,0)(3,2,0)[4]             : AIC=3161.315, Time=0.32 sec
 ARIMA(0,0,0)(3,2,1)[4]             : AIC=inf, Time=1.92 sec
 ARIMA(0,0,0)(2,2,1)[4]             : AIC=inf, Time=0.79 sec
 ARIMA(0,0,1)(3,2,0)[4]             : AIC=3161.698, Time=2.08 sec
 ARIMA(1,0,1)(3,2,0)[4]             : AIC=3167.076, Time=3.19 sec
 ARIMA(0,0,0)(3,2,0)[4] intercept   : AIC=3163.157, Time=2.36 sec

Best model:  ARIMA(0,0,0)(3,2,0)[4]
Total fit time: 20.648 seconds
```

Double-click (or enter) to edit

```
results_officesup=arimamodel2(df_third)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,2,0)[4]             : AIC=inf, Time=0.32 sec
 ARIMA(0,0,0)(0,2,0)[4]             : AIC=3201.249, Time=0.01 sec
 ARIMA(1,0,0)(1,2,0)[4]             : AIC=3144.586, Time=0.18 sec
 ARIMA(0,0,1)(0,2,1)[4]             : AIC=inf, Time=0.44 sec
 ARIMA(1,0,0)(0,2,0)[4]             : AIC=3202.852, Time=0.03 sec
 ARIMA(1,0,0)(2,2,0)[4]             : AIC=3088.745, Time=0.73 sec
 ARIMA(1,0,0)(3,2,0)[4]             : AIC=3064.551, Time=1.68 sec
 ARIMA(1,0,0)(3,2,1)[4]             : AIC=inf, Time=2.90 sec
 ARIMA(1,0,0)(2,2,1)[4]             : AIC=inf, Time=0.93 sec
 ARIMA(0,0,0)(3,2,0)[4]             : AIC=3065.620, Time=0.38 sec
 ARIMA(2,0,0)(3,2,0)[4]             : AIC=3066.541, Time=2.12 sec
 ARIMA(1,0,1)(3,2,0)[4]             : AIC=inf, Time=1.91 sec
 ARIMA(0,0,1)(3,2,0)[4]             : AIC=3064.416, Time=0.74 sec
 ARIMA(0,0,1)(2,2,0)[4]             : AIC=3088.597, Time=0.38 sec
 ARIMA(0,0,1)(3,2,1)[4]             : AIC=inf, Time=1.45 sec
 ARIMA(0,0,1)(2,2,1)[4]             : AIC=inf, Time=0.49 sec
 ARIMA(0,0,2)(3,2,0)[4]             : AIC=3066.079, Time=1.18 sec
 ARIMA(1,0,2)(3,2,0)[4]             : AIC=inf, Time=1.98 sec
 ARIMA(0,0,1)(3,2,0)[4] intercept   : AIC=3066.406, Time=0.86 sec
```

```
Best model:  ARIMA(0,0,1)(3,2,0)[4]
Total fit time: 18.720 seconds
```

```
#results_4p.order
```

```
results_furniture=arimamodel(s)#Model _Selectio with Season repeating after 4
```

```
Performing stepwise search to minimize aic
 ARIMA(1,2,1)(0,1,0)[12]             : AIC=inf, Time=0.10 sec
 ARIMA(0,2,0)(0,1,0)[12]             : AIC=591.823, Time=0.02 sec
 ARIMA(1,2,0)(1,1,0)[12]             : AIC=569.514, Time=0.08 sec
 ARIMA(0,2,1)(0,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,2,0)(0,1,0)[12]             : AIC=573.839, Time=0.03 sec
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py:522: ModelFit
Traceback:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py", lin
    fit.fit(y, X=X, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 597, i
    self._fit(y, X, **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 518, i
    fit, self.arima_res_ = _fit_wrapper()
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 512, i
    **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/mlemodel.py
    @loglikelihood_burn.setter
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
    warning_description = ' for %s' % warning_description
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py", line 40
    else:
ValueError: maxlag should be < nobs

  warnings.warn(warning_str, ModelFitWarning)
 ARIMA(1,2,0)(2,1,0)[12]             : AIC=571.172, Time=0.32 sec
 ARIMA(1,2,0)(1,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,2,0)(0,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(1,2,0)(2,1,1)[12]             : AIC=inf, Time=nan sec
 ARIMA(0,2,0)(1,1,0)[12]             : AIC=590.385, Time=0.12 sec
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py:522: ModelFit
Traceback:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py", lin
    fit.fit(y, X=X, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 597, i
    self._fit(y, X, **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 518, i
    fit, self.arima_res_ = _fit_wrapper()
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/arima.py", line 512, i
    **fit_args)
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/mlemodel.py
```

```
    @loglikelihood_burn.setter
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py
    warning_description = ' for %s' % warning_description
  File "/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py", line 40
    else:
ValueError: maxlag should be < nobs

  warnings.warn(warning_str, ModelFitWarning)
/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py:522: ModelFit
Traceback:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/pmdarima/arima/_auto_solvers.py", lin
```

```
result_tech2=arimamodel(df_second)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,1,0)[12] intercept   : AIC=3108.267, Time=0.08 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=3106.248, Time=0.02 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=3066.371, Time=0.28 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=1.64 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=3104.910, Time=0.03 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=3107.511, Time=0.04 sec
 ARIMA(1,0,0)(2,1,0)[12] intercept   : AIC=3047.545, Time=2.14 sec
 ARIMA(1,0,0)(3,1,0)[12] intercept   : AIC=3059.166, Time=1.44 sec
 ARIMA(1,0,0)(2,1,1)[12] intercept   : AIC=3040.772, Time=5.32 sec
 ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=3035.800, Time=2.14 sec
 ARIMA(1,0,0)(0,1,1)[12] intercept   : AIC=inf, Time=1.40 sec
 ARIMA(1,0,0)(1,1,2)[12] intercept   : AIC=inf, Time=6.89 sec
 ARIMA(1,0,0)(0,1,2)[12] intercept   : AIC=3053.443, Time=0.82 sec
 ARIMA(1,0,0)(2,1,2)[12] intercept   : AIC=3056.056, Time=1.09 sec
 ARIMA(0,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.68 sec
 ARIMA(2,0,0)(1,1,1)[12] intercept   : AIC=3055.193, Time=0.48 sec
 ARIMA(1,0,1)(1,1,1)[12] intercept   : AIC=3054.953, Time=0.73 sec
 ARIMA(0,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=1.91 sec
 ARIMA(2,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=2.58 sec
 ARIMA(1,0,0)(1,1,1)[12]             : AIC=3032.149, Time=1.53 sec
 ARIMA(1,0,0)(0,1,1)[12]             : AIC=inf, Time=0.73 sec
 ARIMA(1,0,0)(1,1,0)[12]             : AIC=3065.572, Time=0.21 sec
 ARIMA(1,0,0)(2,1,1)[12]             : AIC=inf, Time=2.90 sec
 ARIMA(1,0,0)(1,1,2)[12]             : AIC=3054.051, Time=1.12 sec
 ARIMA(1,0,0)(0,1,0)[12]             : AIC=3106.073, Time=0.03 sec
 ARIMA(1,0,0)(0,1,2)[12]             : AIC=3052.233, Time=0.70 sec
 ARIMA(1,0,0)(2,1,0)[12]             : AIC=3047.516, Time=1.81 sec
 ARIMA(1,0,0)(2,1,2)[12]             : AIC=inf, Time=5.29 sec
 ARIMA(0,0,0)(1,1,1)[12]             : AIC=3037.475, Time=0.26 sec
 ARIMA(2,0,0)(1,1,1)[12]             : AIC=3054.159, Time=0.48 sec
 ARIMA(1,0,1)(1,1,1)[12]             : AIC=inf, Time=1.94 sec
 ARIMA(0,0,1)(1,1,1)[12]             : AIC=inf, Time=1.07 sec
 ARIMA(2,0,1)(1,1,1)[12]             : AIC=3027.171, Time=2.17 sec
 ARIMA(2,0,1)(0,1,1)[12]             : AIC=inf, Time=2.86 sec
 ARIMA(2,0,1)(1,1,0)[12]             : AIC=3055.820, Time=2.52 sec
 ARIMA(2,0,1)(2,1,1)[12]             : AIC=inf, Time=5.65 sec
 ARIMA(2,0,1)(1,1,2)[12]             : AIC=3052.761, Time=3.45 sec
```

```
 ARIMA(2,0,1)(0,1,0)[12]             : AIC=3108.648, Time=0.16 sec
 ARIMA(2,0,1)(0,1,2)[12]             : AIC=3027.430, Time=4.49 sec
 ARIMA(2,0,1)(2,1,0)[12]             : AIC=3058.348, Time=2.06 sec
 ARIMA(2,0,1)(2,1,2)[12]             : AIC=3053.963, Time=7.41 sec
 ARIMA(3,0,1)(1,1,1)[12]             : AIC=3027.128, Time=3.03 sec
 ARIMA(3,0,1)(0,1,1)[12]             : AIC=3052.957, Time=0.72 sec
 ARIMA(3,0,1)(1,1,0)[12]             : AIC=3065.356, Time=0.84 sec
 ARIMA(3,0,1)(2,1,1)[12]             : AIC=inf, Time=7.23 sec
 ARIMA(3,0,1)(1,1,2)[12]             : AIC=3053.733, Time=3.45 sec
 ARIMA(3,0,1)(0,1,0)[12]             : AIC=3108.464, Time=0.18 sec
 ARIMA(3,0,1)(0,1,2)[12]             : AIC=inf, Time=5.57 sec
 ARIMA(3,0,1)(2,1,0)[12]             : AIC=3059.354, Time=2.39 sec
 ARIMA(3,0,1)(2,1,2)[12]             : AIC=3052.527, Time=7.77 sec
 ARIMA(3,0,0)(1,1,1)[12]             : AIC=inf, Time=1.79 sec
 ARIMA(3,0,2)(1,1,1)[12]             : AIC=3054.611, Time=4.32 sec
 ARIMA(2,0,2)(1,1,1)[12]             : AIC=inf, Time=3.20 sec
 ARIMA(3,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=3.25 sec

Best model:  ARIMA(3,0,1)(1,1,1)[12]
```

```
result_tech2.order
```

```
(3, 0, 1)
```

```
p_tech=result_tech2.order
q_tech=result_tech2.seasonal_order
```

```
print(p_tech,q_tech)
```

```
(3, 0, 1) (1, 1, 1, 12)
```

```
p1=results_furniture.order
q1=results_furniture.seasonal_order
```

```
print(p1,q1)
```

```
(3, 2, 0) (1, 1, 0, 12)
```

```
p_office=results_officesup.order
q_office=result_tech2.seasonal_order
```

```
print(p_office)
```

```
(0, 0, 1)
```

```
second_best_tech=[3,0,2]
second_best_off=[1,0,2]
second_best_furn=[2,0,1]
```

```
model_storage={"Furniture":[second_best_furn],"office":[second_best_off],"Technology":[second
```

```
model_storage["Furniture"][0]
```

```
[2, 0, 1]
```

```
#Functions to call Furniture
def recall(Product):
  r=model_storage[Product]
  return r[0]
```

```
recall("Furniture")
```

```
[2, 0, 1]
```

```
print(p1,"Furniture Best Mode SARIMAX")
print(q1,"Furniture Best Model SARIMAX")
```

```
(3, 2, 0) Furniture Best Mode SARIMAX
(1, 1, 0, 12) Furniture Best Model SARIMAX
```

```
best_model = SARIMAX(df_first, order=p1, seasonal_order=q1).fit()
print(best_model.summary())
```

```
                                 Statespace Model Results
==========================================================================================
Dep. Variable:                  Monthly_Quantity   No. Observations:
Model:             SARIMAX(3, 2, 0)x(1, 1, 0, 12)   Log Likelihood                 -274.(
Date:                           Sat, 30 Apr 2022   AIC                             558.(
Time:                                   20:51:06   BIC                             564.7
Sample:                               01-01-2011   HQIC                            560.(
                                    - 06-01-2014
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.3336      0.277     -4.809      0.000      -1.877      -0.790
ar.L2         -0.8980      0.310     -2.893      0.004      -1.506      -0.290
ar.L3         -0.3008      0.277     -1.085      0.278      -0.844       0.242
ar.S.L12      -0.5928      0.351     -1.688      0.091      -1.281       0.095
sigma2      2.414e+07   2.46e-09   9.82e+15      0.000    2.41e+07    2.41e+07
===================================================================================
Ljung-Box (Q):                       nan   Jarque-Bera (JB):                  1.24
Prob(Q):                             nan   Prob(JB):                          0.54
Heteroskedasticity (H):             3.49   Skew:                             -0.48
Prob(H) (two-sided):                0.08   Kurtosis:                          3.36
```

```
==================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.49e+32. Star
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
#Auto-Arima Wokring
```

```
autoModel = pmd.auto_arima(df_first, trace=True,
                          error_action='ignore', suppress_warnings=True, seasonal=False)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=803.581, Time=0.12 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=849.236, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=812.043, Time=0.01 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=831.532, Time=0.02 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=802.117, Time=0.04 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=827.007, Time=0.03 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=799.635, Time=0.04 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=801.740, Time=0.04 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=806.529, Time=0.01 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=797.033, Time=0.01 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=795.506, Time=0.01 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=794.002, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=795.112, Time=0.01 sec

Best model:  ARIMA(0,0,0)(0,0,0)[0] intercept
Total fit time: 0.362 seconds
```

```
order = autoModel.order
yhat = list()
model = ARIMA(df_first, order=order)
model_fit = model.fit()
predictions = model_fit.forecast(steps=7)
yhat = yhat + [predictions]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
predictions[0]
```

```
array([5010.88065476, 5010.88065476, 5010.88065476, 5010.88065476,
       5010.88065476, 5010.88065476, 5010.88065476])
```

```
autoModel.order
```

```
(0, 0, 0)
```

```
weekly_result=arimamodel(x_week)
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,1,0)[12] intercept   : AIC=2154.890, Time=0.21 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=2158.147, Time=0.02 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=2102.768, Time=0.65 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=2067.273, Time=0.83 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=2156.149, Time=0.02 sec
 ARIMA(0,0,1)(0,1,0)[12] intercept   : AIC=2158.192, Time=0.13 sec
 ARIMA(0,0,1)(1,1,1)[12] intercept   : AIC=2068.404, Time=1.87 sec
 ARIMA(0,0,1)(0,1,2)[12] intercept   : AIC=2068.468, Time=4.38 sec
 ARIMA(0,0,1)(1,1,0)[12] intercept   : AIC=2102.343, Time=0.46 sec
 ARIMA(0,0,1)(1,1,2)[12] intercept   : AIC=inf, Time=6.42 sec
 ARIMA(0,0,0)(0,1,1)[12] intercept   : AIC=2066.688, Time=0.48 sec
 ARIMA(0,0,0)(1,1,1)[12] intercept   : AIC=2067.793, Time=1.14 sec
 ARIMA(0,0,0)(0,1,2)[12] intercept   : AIC=2067.835, Time=3.29 sec
 ARIMA(0,0,0)(1,1,0)[12] intercept   : AIC=2103.130, Time=0.47 sec
 ARIMA(0,0,0)(1,1,2)[12] intercept   : AIC=inf, Time=4.78 sec
 ARIMA(1,0,0)(0,1,1)[12] intercept   : AIC=2067.527, Time=0.64 sec
 ARIMA(1,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=2.35 sec
 ARIMA(0,0,0)(0,1,1)[12]             : AIC=2064.766, Time=0.25 sec
 ARIMA(0,0,0)(1,1,1)[12]             : AIC=2065.866, Time=0.55 sec
 ARIMA(0,0,0)(0,1,2)[12]             : AIC=2065.908, Time=1.36 sec
 ARIMA(0,0,0)(1,1,0)[12]             : AIC=2101.130, Time=0.14 sec
 ARIMA(0,0,0)(1,1,2)[12]             : AIC=inf, Time=3.51 sec
 ARIMA(1,0,0)(0,1,1)[12]             : AIC=2065.594, Time=0.50 sec
 ARIMA(0,0,1)(0,1,1)[12]             : AIC=2065.339, Time=0.62 sec
 ARIMA(1,0,1)(0,1,1)[12]             : AIC=inf, Time=1.06 sec

Best model:  ARIMA(0,0,0)(0,1,1)[12]
Total fit time: 36.177 seconds
```

```
x_week
```

```
Order Date
2011-01-09    112.124963
2011-01-16    139.525615
2011-01-23     51.239882
2011-01-30     75.593111
2011-02-06     74.928923
                 ...
2014-06-01     98.133120
2014-06-08    137.439680
2014-06-15     65.773500
2014-06-22     59.033909
2014-06-29     71.478385
Freq: W-SUN, Name: Sales, Length: 182, dtype: float64
```

```
#results2.summary()
```

Validation Data Set Preperation for 3 products in US -west

```
valid_west=valid_data[(valid_data['Region']=='West') & (valid_data['Category']=='Furniture')]
valid_west['Synthetic_Date'] = pd.to_datetime(valid_west[['Year', 'Month']].assign(DAY=1))
valid_west.shape
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: 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_

(7, 8)
```

```
valid_Technology=valid_data[(valid_data['Region']=='West') & (valid_data['Category']=='Techno
valid_Technology['Synthetic_Date'] = pd.to_datetime(valid_Technology[['Year', 'Month']].assig
valid_Technology.shape
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: 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_

(7, 8)
```

```
valid_Office_Supplies=valid_data[(valid_data['Region']=='West') & (valid_data['Category']=='O
valid_Office_Supplies['Synthetic_Date'] = pd.to_datetime(valid_Office_Supplies[['Year', 'Mont
valid_Office_Supplies.shape
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: 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_

(7, 8)
```

```
df_valid_office_supply=valid_Office_Supplies.iloc[:,6:8]
df_valid_office_supply
```

```
df_valid_technology=valid_Technology.iloc[:,6:8]
```

```
df_valid_furniture=valid_west.iloc[:,6:8]
```

df_valid

```
history=[x for x in s]
history2=[x for x in s2_tech]
history3=[x for x in s3_office]
```

```
train=df_first.values
#train
```

```
df_valid_furniture.index=df_valid_furniture['Synthetic_Date']
del (df_valid_furniture['Synthetic_Date'])
```

```
df_valid_furniture
```

```
df_valid_technology.index=df_valid_technology['Synthetic_Date']
```

```
df_valid_technology
del(df_valid_technology['Synthetic_Date'])
```

```
df_valid_technology
```

```
df_valid_office_supply.index=df_valid_office_supply['Synthetic_Date']
del df_valid_office_supply['Synthetic_Date']
```

```
valid_furniture=df_valid_furniture.values
```

```
df_valid_office_supply
```

```
valid_furniture=valid_furniture.reshape(-1,)
```

```
valid_furniture
```

```
array([ 997.7  , 6156.531, 4596.342, 9531.501, 5954.845, 8363.685,
       6555.152])
```

```
df_valid_furniture.index
```

```
DatetimeIndex(['2014-06-01', '2014-07-01', '2014-08-01', '2014-09-01',
               '2014-10-01', '2014-11-01', '2014-12-01'],
              dtype='datetime64[ns]', name='Synthetic_Date', freq=None)
```

```
df_valid_furniture
```

```
df_valid2=df_valid_furniture.values.reshape(-1,)
```

```
df_valid2
```

```
array([ 997.7  , 6156.531, 4596.342, 9531.501, 5954.845, 8363.685,
       6555.152])
```

```
history
```

```
[1674.203,
 202.888,
 3496.594,
 2965.638,
 1955.602,
 2032.822,
 5836.966,
```

```
      3830.808,
      6603.567,
      4470.3,
      5102.0585,
      11911.0015,
      11172.698,
      2944.6569999999997,
      6845.691,
      1115.496,
      4954.8785,
      2580.674,
      5015.911,
      3529.011,
      5371.75,
      4263.827,
      5262.546,
      3947.804,
      4199.63,
      243.548,
      3813.182,
      8063.847,
      5738.4710000000005,
      6359.0470000000005,
      2624.822,
      7763.148,
      9438.6185,
      2928.643,
      9244.87,
      13401.814,
      1804.413,
      6922.5419999999995,
      5467.3,
      6588.428,
      5135.372,
      3631.9005]
```

```
run_type=['model1','model2','model3','model4']
def running(run_type):
  predictions=list()
  if run_type=='model1' :
   for t in range(len(df_valid2)):#Walk through the Array of Future Values
    model=ARIMA(history,order=(0,1,1))#Training through History object
    model_fit=model.fit()
    output=model_fit.forecast()
    yhat=output[0][0]
    predictions.append(yhat)
    obs=valid_furniture[t]
    history.append(obs)
  elif run_type=='model2':
   model=ARIMA(df_first,order=(1,0,0))#training Data
   model_fit=model.fit()
   output=model_fit.forecast(steps=7)
   predictions=output[0]
```

```
  elif run_type=='model3':
    model=SARIMAX(df_first,order=p1,seasonal_order=q1).fit()
    output=model.get_forecast(steps=7,dynamic=True)
    predictions=output.predicted_mean
  return predictions
```

```
model=ARIMA(df_first,order=(1,0,0))
model_fit=model.fit()
output=model_fit.forecast(steps=7)[0]
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
preditions3=running('model3')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
df_valid2
history
```

```
[1674.203,
 202.888,
 3496.594,
 2965.638,
 1955.602,
 2032.822,
 5836.966,
 3830.808,
 6603.567,
 4470.3,
 5102.0585,
 11911.0015,
 11172.698,
 2944.6569999999997,
 6845.691,
 1115.496,
 4954.8785,
 2580.674,
 5015.911,
 3529.011,
 5371.75,
 4263.827,
 5262.546,
 3947.804,
 4199.63,
 243.548,
```

```
 3813.182,
 8063.847,
 5738.4710000000005,
 6359.0470000000005,
 2624.822,
 7763.148,
 9438.6185,
 2928.643,
 9244.87,
 13401.814,
 1804.413,
 6922.5419999999995,
 5467.3,
 6588.428,
 5135.372,
 3631.9005]
```

```
predictions=list()
for t in range(len(df_valid2)):
    model=ARIMA(history,order=(0,1,1))
    model_fit=model.fit()
    output=model_fit.forecast()
    yhat=output[0][0]
    print(yhat)
    predictions.append(yhat)
    obs=df_valid2[t]
    history.append(obs)
```

```
6599.211497642142
4832.600237184487
6144.327614172404
6093.43797563033
6341.355011274371
6440.6618630326875
6611.984878125183
```

```
predictions
```

```
[6599.211497642142,
 4832.600237184487,
 6144.327614172404,
 6093.43797563033,
 6341.355011274371,
 6440.6618630326875,
 6611.984878125183]
```

```
model=SARIMAX(df_first,order=p1,seasonal_order=q1).fit()
output=model.get_forecast(steps=7,dynamic=True)
predictions=output.predicted_mean
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
```

```
  % freq, ValueWarning)
```

```
history_tech=[x for x in s2_tech]
history_off=[x for x in s3_office ]
```

```
history[0]
```

```
1674.203
```

VALIDATION TESTING FOR FURNITURE

```
# run_type=['model1','model2','model3','model4']
# def validation_test(run_type,product_valid,history,X_TR,product_type) :
#   predictions=list()
#   train=X_TR.values
#   product_valid=product_valid
#   if product_type=='Furniture':
#     train=X_TR.values
#     print(len(product_valid))
#     if run_type=='model1' :
#      for t in range(len(product_valid)):
#        model=ARIMA(history,order=(1,1,0))
#        model_fit=model.fit()
#        output=model_fit.forecast()
#        yhat=output[0]
#        predictions.append(yhat)
#        obs=product_valid.values[t]
#        history.append(obs)
#     elif run_type=='model2':
#       model=SARIMAX(train,order=(1,0,0)).fit()
#       output=model.get_forecast(steps=7,dynamic=True)
#       predictions=output.predicted_mean
#     elif run_type=='model3':
#       model=SARIMAX(train,order=p1,seasonal_order=q1).fit()
#       output=model.get_forecast(steps=7,dynamic=True)
#       predictions=output.predicted_mean
#   elif product_type=='Technology':
#      if run_type=='model1' :
#       for t in range(len(product_valid)):
#         model=ARIMA(history_tech,order=(1,1,0))
#         model_fit=model.fit()
#         output=model_fit.forecast()
#         yhat=output[0]
```

```
#         predictions.append(yhat)
#         obs=product_valid[t]
#         history2.append(obs)
#      elif run_type=='model2':
#       model=SARIMAX(train,order=second_best_tech).fit()
#       output=model.get_forecast(steps=7,dynamic=True)
#       predictions=output.predicted_mean
#      elif run_type=='model3':
#       model=SARIMAX(train,order=p_tech,seasonal_order=q_tech).fit()
#       output=model.get_forecast(steps=7,dynamic=True)
#       predictions=output.predicted_mean
#   elif product_type=="Office Supplies":
#      if run_type=='model1' :
#       for t in range(len(product_valid)):
#         model=ARIMA(history3,order=(1,1,0))
#         model_fit=model.fit()
#         output=model_fit.forecast()
#         yhat=output[0]
#         predictions.append(yhat)
#         obs=product_valid[t]
#         history2.append(obs)
#      elif run_type=='model2':
#       model=SARIMAX(train,order=second_best_off).fit()
#       output=model.get_forecast(steps=7,dynamic=True)
#       predictions=output.predicted_mean
#      elif run_type=='model3':
#       model=SARIMAX(train,order=p_office,seasonal_order=q_office).fit()
#       output=model.get_forecast(steps=7,dynamic=True)
#       predictions=output.predicted_mean
#   return predictions
```

```
#validation_test(run_type='model1',product_valid=df_valid_furniture,history=history,X_TR=df_f
```

```
model=ARIMA(df_first,order=(1,1,0))
model_fit=model.fit()
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
#Creating models by Different Products
```

```
model_fit.plot_predict(dynamic=False)#
plt.title("Forecast -ARIMA[1,1,0] for West US Market Furniture")
plt.show()
```

```
predictions
```

```
2014-07-01    2553.536198
2014-08-01    3123.251212
2014-09-01    3630.314998
2014-10-01    -236.511177
2014-11-01    1877.687111
2014-12-01    1947.568999
2015-01-01   -3516.479009
Freq: MS, dtype: float64
```

```
history= [x for x in s]
```

```
predictions_furniture_model1 =running(run_type='model1')
```

```
predictions_furniture_model1
```

```
[6599.211497642142,
 4832.600237184487,
 6144.327614172404,
 6093.43797563033,
 6341.355011274371,
 6440.6618630326875,
 6611.984878125183]
```

```
#approach 2
predictions_furniture_model2=running(run_type='model2')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
predictions_furniture_model3=running(run_type='model3')
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
# Optionally we can use SARIMAX Library which can be set as in sample and out of sample using
model33 = SARIMAX(df_first, order=(1,0,0), trend='c')
results = model33.fit()
# Make in-sample prediction
forecast = results.get_prediction(start=-12)
mean_forecast=forecast.predicted_mean
# The below represents in sample values which means we predict 1 value  and then use its true
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
#plt.plot(mean_forecast.values,color='red',label='forecast')
forecast=results.get_prediction(start = len(df_first)-5,dynamic= True)
```

```
p=forecast.predicted_mean
p
```

```
2014-02-01    4597.371835
2014-03-01    4966.138018
2014-04-01    5014.827778
2014-05-01    5021.256493
2014-06-01    5022.105304
Freq: MS, dtype: float64
```

```
df_first.plot(legend = True)
p.plot(legend=True,label='Forecast')
plt.title("ARIMA -1,0,0 for US Market Furniture(Best Training Model) ")
plt.show()
```

```
fig, ax = plt.subplots()
df_first.plot(legend = True,ax=ax)
p.plot(legend=True,ax=ax)
ax.legend(["True", "Forecast"]);
ax.set_title("ARIMA 110 for US West Furnuture ")
```

```
#results.get_predictions(start = len(df_train), end = len(df) - 1, type = 'levels').rename('S
```

```
#approach 1
#predictions_walkover=running(run_type='model1')
```

```
#approach2
#predictions_2=running(run_type='model2')
```

```
plt.plot(df_valid_furniture.index,predictions_furniture_model1,color='red')
plt.plot(df_valid_furniture.index,predictions_furniture_model2,color='blue')
```

```
y_to_train = df_first[:'2014-01-01'] # dataset to train
valid_y=df_first['2014-01-01':]
planning_pd=len(valid_y)
print(planning_pd)
model=ARIMA(y_to_train,order=(0,1,0))
model_fit=model.fit()

output=model_fit.forecast(planning_pd)
```

```
6
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
```

```
training_data.groupby
```

```
<bound method DataFrame.groupby of            Row ID        Order ID  Order Date
Order Date
2011-01-01   42433     AG-2011-2040  2011-01-01    6/1/2011  Standard Class
2011-01-01   22253    IN-2011-47883  2011-01-01    8/1/2011  Standard Class
2011-01-01   48883     HU-2011-1220  2011-01-01    5/1/2011    Second Class
2011-01-01   11731  IT-2011-3647632  2011-01-01    5/1/2011    Second Class
2011-01-01   22255    IN-2011-47883  2011-01-01    8/1/2011  Standard Class
...            ...              ...         ...         ...             ...
2013-12-31   42653     TU-2013-9400  2013-12-31    4/1/2014  Standard Class
2013-12-31   39963   CA-2013-163951  2013-12-31    3/1/2014     First Class
2013-12-31   37057   US-2013-111528  2013-12-31  31-12-2013        Same Day
2013-12-31   36058   CA-2013-117660  2013-12-31    5/1/2014  Standard Class
2013-12-31   49701     IZ-2013-2550  2013-12-31    2/1/2014    Second Class

           Customer ID    Customer Name      Segment         City  \
Order Date
2011-01-01    TB-11280  Toby Braunhardt     Consumer   Constantine
2011-01-01    JH-15985      Joseph Holt     Consumer   Wagga Wagga
2011-01-01      AT-735    Annie Thurman     Consumer      Budapest
```

```
2011-01-01    EM-14140     Eugene Moren  Home Office     Stockholm
2011-01-01    JH-15985      Joseph Holt     Consumer   Wagga Wagga
...                ...              ...          ...           ...
2013-12-31    TM-11490    Tony Molinari     Consumer     Gaziantep
2013-12-31    CJ-11875     Carl Jackson    Corporate  Philadelphia
2013-12-31    JP-16135  Julie Prescott  Home Office   Los Angeles
2013-12-31    BM-11785      Bryan Mills     Consumer      Columbus
2013-12-31     DM-3015    Darrin Martin     Consumer         Basra

                      State  ...    Sales  Quantity Discount    Profit  \
Order Date                   ...
2011-01-01      Constantine  ...  408.300         2      0.0  106.1400
2011-01-01  New South Wales  ...  120.366         3      0.1   36.0360
2011-01-01         Budapest  ...   66.120         4      0.0   29.6400
2011-01-01        Stockholm  ...   44.865         3      0.5  -26.0550
2011-01-01  New South Wales  ...  113.670         5      0.1   37.7700
...                     ...  ...      ...       ...      ...       ...
2013-12-31        Gaziantep  ...   10.080         1      0.6   -5.5500
2013-12-31     Pennsylvania  ...   16.520         5      0.2    1.6520
2013-12-31       California  ...    6.384         1      0.2    2.1546
2013-12-31             Ohio  ...    5.904         2      0.2    1.9926
2013-12-31        Al Basrah  ...   13.020         1      0.0    4.0200

           Shipping Cost Order Priority             Order_Date_c  Year  \
Order Date
2011-01-01         35.46         Medium 2011-01-01 00:00:00+00:00  2011
2011-01-01          9.72         Medium 2011-01-01 00:00:00+00:00  2011
2011-01-01          8.17           High 2011-01-01 00:00:00+00:00  2011
2011-01-01          4.82           High 2011-01-01 00:00:00+00:00  2011
2011-01-01          4.70         Medium 2011-01-01 00:00:00+00:00  2011
...                  ...            ...                       ...   ...
2013-12-31          0.59         Medium 2013-12-31 00:00:00+00:00  2013
2013-12-31          0.42           High 2013-12-31 00:00:00+00:00  2013
2013-12-31          0.34         Medium 2013-12-31 00:00:00+00:00  2013
2013-12-31          0.23         Medium 2013-12-31 00:00:00+00:00  2013
2013-12-31          0.14         Medium 2013-12-31 00:00:00+00:00  2013
```

```
y_to_train.head(3)
```

```
df_first.plot(marker='o', color='black', legend=True, figsize=(14, 7))
#model_fit.predicted_mean.plot(marker="o", color='blue')
pd.DataFrame(output[0]).plot(marker='o', color='blue', legend=True)
```

```
output[0]
```

```
array([1808.02994444, 1811.64688889, 1815.26383333, 1818.88077778,
       1822.49772222, 1826.11466667])
```

```
predictions_furniture_model3
```

```
2014-07-01    2553.536198
2014-08-01    3123.251212
2014-09-01    3630.314998
2014-10-01    -236.511177
2014-11-01    1877.687111
```

```
2014-12-01    1947.568999
2015-01-01   -3516.479009
Freq: MS, dtype: float64
```

```
def rmse(yhat,y_actual):
     p=np.subtract(y_actual,yhat)
     magn=np.linalg.norm(p)#L2 NORM OF VECOT
     q=magn
     mse2=q/len(y_actual)
     return mse2

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

```
DF=pd.DataFrame(predictions_furniture_model1)
yhat_vector=DF.values.reshape(-1,)
```

```
valid_furniture
```

```
array([ 997.7  , 6156.531, 4596.342, 9531.501, 5954.845, 8363.685,
       6555.152])
```

```
# def single_array(valid):
#   valid_new=[]
#   for i in range(len(predictions_2)):#same lenght as predictions
#     valid_new.append(valid[i][0])
#   return valid_new

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

```
valid_furniture
```



```
array([ 997.7  , 6156.531, 4596.342, 9531.501, 5954.845, 8363.685,
       6555.152])
```



```
mape_furniture_model1=test_evaluation(predictions_furniture_model1,valid_furniture)# Both mus
```

```
mape_furniture_model2=test_evaluation(predictions_furniture_model2,valid_furniture)
```

```
mape_furniture_model3=test_evaluation(predictions_furniture_model3,valid_furniture)
```

```
mape_furniture_model3
list_m=[mape_furniture_model1,mape_furniture_model2,mape_furniture_model3]
x=['model-110 ARIMA ','model: 100 ARIMA','model:3,2,0 SARIMAX']


plt.bar(x,list_m)
plt.title('ARIMA model_comparison -Validation_Set')
```

Model 110 will be the best mode for Furniture across the globe/regions

```
second_best_tech
```

```
[3, 0, 2]
```

```
run_type=['model1','model2','model3','model4']
def validation_test(run_type,product_valid,history,X_TR,product_type) :
  predictions=list()
  train=X_TR.values#array
  product_valid=product_valid.values#array
  if product_type=='Furniture':
    train=X_TR.values
    print(len(product_valid))
    if run_type=='model1' :
     for t in range(len(product_valid)):
        model=ARIMA(history,order=(1,1,0))#common
        model_fit=model.fit()
        output=model_fit.forecast()
        yhat=output[0][0]
        predictions.append(yhat)
        obs=product_valid[t]
```

```
      history.append(obs)
    elif run_type=='model2':
     model=SARIMAX(train,order=(1,0,0)).fit()
     output=model.get_forecast(steps=7,dynamic=True)
     predictions=output.predicted_mean
    elif run_type=='model3':
     model=SARIMAX(train,order=p1,seasonal_order=q1).fit()
     output=model.get_forecast(steps=7,dynamic=True)
     predictions=output.predicted_mean
  elif product_type=='Technology':
     if run_type=='model1' :
      for t in range(len(product_valid)):
        model=ARIMA(history_tech,order=(1,1,0))
        model_fit=model.fit()
        output=model_fit.forecast()
        yhat=output[0][0]
        predictions.append(yhat)
        obs=product_valid[t]
        history2.append(obs)
     elif run_type=='model2':
      model=SARIMAX(train,order=second_best_tech).fit()
      output=model.get_forecast(steps=7,dynamic=True)
      predictions=output.predicted_mean
     elif run_type=='model3':
      model=SARIMAX(train,order=p_tech,seasonal_order=q_tech).fit()
      output=model.get_forecast(steps=7,dynamic=True)
      predictions=output.predicted_mean
  elif product_type=="Office Supplies":
     if run_type=='model1' :
      for t in range(len(product_valid)):
        model=ARIMA(history3,order=(1,1,0))
        model_fit=model.fit()
        output=model_fit.forecast()
        yhat=output[0][0]
        predictions.append(yhat)
        obs=product_valid[t]
        history2.append(obs)
     elif run_type=='model2':
      model=SARIMAX(train,order=second_best_off).fit()
      output=model.get_forecast(steps=7,dynamic=True)
      predictions=output.predicted_mean
     elif run_type=='model3':
      model=SARIMAX(train,order=p_office,seasonal_order=q_office).fit()
      output=model.get_forecast(steps=7,dynamic=True)
      predictions=output.predicted_mean
  return predictions
```

```
yhat_model1_furniture=validation_test(run_type='model1',product_valid=df_valid_furniture,hist
```

```
7
/usr/local/lib/python3.7/dist-packages/statsmodels/base/data.py:629: VisibleDeprecationW
  exog_names = ['x%d' % i for i in range(1, exog.shape[1])]
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/arima_model.py:424: VisibleDeprec
```

```
#yhat_model3_furniture
```

```
#test_evaluation(yhat_model3_furniture,df_valid_furniture.values)
```

```
yhat_model2_furniture=validation_test(run_type='model2',product_valid=df_valid_furniture,hist
```

```
7
```

```
yhat_model3_furniture=validation_test(run_type='model3',product_valid=df_valid_furniture,hist
```

```
7
```

```
yhat_model1_tech=validation_test(run_type='model1',product_valid=df_valid_technology,history=
yhat_model2_tech=validation_test(run_type='model2',product_valid=df_valid_technology,history=
```

```
yhat_model1_office=validation_test(run_type='model1',product_valid=df_valid_office_supply,his
yhat_model2_office=validation_test(run_type='model2',product_valid=df_valid_office_supply,his
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/statespace/sarimax.py:949: UserWa
  params_exog = []
```

```
yhat_model3_office=validation_test(run_type='model3',product_valid=df_valid_office_supply,his
```

```
df_valid_technology
```

```
yhat_mode3_tech=validation_test(run_type='model3',product_valid=df_valid_technology,history=h
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/base/model.py:512: ConvergenceWarning
```

```
df_valid_technology
```

```
mape_office1=test_evaluation(yhat_model1_office,df_valid_office_supply.values)
mape_office2=test_evaluation(yhat_model2_office,df_valid_office_supply.values)
mape_office3=test_evaluation(yhat_model3_office,df_valid_office_supply.values)
```

```
mape_tech_1=test_evaluation(yhat_model1_tech,df_valid_technology.values)
mape_tech_2=test_evaluation(yhat_model2_tech,df_valid_technology.values)
```

```
list_m=[mape_tech_1,mape_tech_2]
x=['model-110 ARIMA ','Model -3,2,0 ARIMA']
plt.bar(x,list_m)
plt.title('MAPE Score  -Validation_Set_Tech Products ')
```

```
results_officesup
```

```
    ARIMA(order=(0, 0, 1), scoring_args={}, seasonal_order=(3, 2, 0, 4),
          suppress_warnings=True, with_intercept=False)
```

```
list_m=[mape_office1,mape_office2,mape_office3]
x=['model-110 ARIMA ','Model -1,0,2 ARIMA','Model--001[3,2,0,4]']
plt.bar(x,list_m)
plt.title('MAPE Score  -Validation_Set_Office Products ')
```

Office Supply Validation Data Set

```
# We are trying  001 models on
```

```
# model_pipeline=['model1','model2','model3']
# def test_complete(model_type,valid2):
#    predictions=running(run_type=model_type)
#    mape=test_evaluation(predictions,valid2)
#    return mape
# x=defaultdict()
# for model in model_pipeline:
#     mape_model=test_complete(model,valid_furniture)
#     x[model]=mape_model
```

```
x
```

```
['model-110 ARIMA ', 'Model -1,0,2 ARIMA', 'Model--001[3,2,0,4]']
```

```
# valid_new3
```

```
#r=rmse(yhat_vector,valid)

#q=MAPE(,valid_new3)
#print("rMSE error is : %3d, MAPE erros is  : %2d for 5 step process " % (r,q))
```

```
x_daily=cohort_1.resample('D').mean()
```

```
daily=x_daily['Sales']
```

```
daily_interp=daily.interpolate(limit=2, limit_direction="forward")
```

```
plt.plot(daily_interp)
```

```
df_first.plot(marker='o', color='black', legend=True, figsize=(14, 7))
```

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

```
imp_mean
```

```
SimpleImputer()
```

```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(daily_interp, lags=200, zero=False, ax=ax1)
ax2 = fig.add_subplot(212)
```

```
Market_List
```

```
array(['Africa', 'APAC', 'EMEA', 'EU', 'US', 'LATAM', 'Canada'],
      dtype=object)
```

```
from collections import defaultdict
cohort={}
series_size=defaultdict()
for i in range(len(Market_List)):
     cohort[i]=training_data2[(training_data2['Market']==Market_List[i]) & (training_data2['
     p=len(cohort[i])
     c=Market_List[i]
     series_size[c]=(p)
```

```
series_size
```

```
defaultdict(None,
            {'APAC': 4951,
             'Africa': 2458,
             'Canada': 218,
             'EMEA': 2626,
             'EU': 5201,
             'LATAM': 4731,
             'US': 4888})
```

```
series_size['APAC']
```

```
4951
```

```
cohort
```

```
{0:             Row ID      Order ID  Ship Date       Ship Mode Customer ID  \
 Order Date
 2011-01-01   42433  AG-2011-2040   6/1/2011  Standard Class    TB-11280
 2011-01-02   44508  AO-2011-1390   4/2/2011    Second Class     DK-3150
 2011-01-03   50129   NI-2011-190   6/3/2011  Standard Class     EH-3765
 2011-01-06   44800  SO-2011-3360   4/6/2011    Second Class      AJ-945
 2011-01-06   44799  SO-2011-3360   4/6/2011    Second Class      AJ-945
```

```
...          ...   ...           ...         ...             ...
2013-10-31   45831  NI-2013-3460  5/11/2013    Second Class    MV-8190
2011-12-31   41681  CG-2011-8620   2/1/2012    Second Class    NB-8655
2013-12-31   47215  AO-2013-6910   3/1/2014     First Class    JH-5985
2013-12-31   48725   SF-2013-680   6/1/2014  Standard Class    NS-8505
2013-12-31   48724   SF-2013-680   6/1/2014  Standard Class    NS-8505

              Customner Name    Segment         City        State  \
Order Date
2011-01-01  Toby Braunhardt   Consumer  Constantine  Constantine
2011-01-02   David Kendrick  Corporate       Luanda       Luanda
2011-01-03     Edward Hooks  Corporate         Kano         Kano
2011-01-06    Ashley Jarboe   Consumer    Mogadishu     Banaadir
2011-01-06    Ashley Jarboe   Consumer    Mogadishu     Banaadir
...                    ...        ...          ...          ...
2013-10-31   Mike Vittorini   Consumer        Lagos        Lagos
2011-12-31        Nona Balk  Corporate       Likasi      Katanga
2013-12-31      Joseph Holt   Consumer       Luanda       Luanda
2013-12-31  Neola Schneider   Consumer     Pretoria      Gauteng
2013-12-31  Neola Schneider   Consumer     Pretoria      Gauteng

                                      Country  ...    Sales Quantity Discount  \
Order Date                                     ...
2011-01-01                            Algeria  ...  408.300        2      0.0
2011-01-02                             Angola  ...  206.400        1      0.0
2011-01-03                            Nigeria  ...   25.317        1      0.7
2011-01-06                            Somalia  ...   21.180        2      0.0
2011-01-06                            Somalia  ...   10.860        1      0.0
...                                       ...  ...      ...      ...      ...
2013-10-31                            Nigeria  ...    2.925        1      0.7
2011-12-31  Democratic Republic of the Congo  ...   22.410        1      0.0
2013-12-31                             Angola  ...  848.400        4      0.0
2013-12-31                       South Africa  ...   81.780        1      0.0
2013-12-31                       South Africa  ...   17.880        1      0.0

             Profit Shipping Cost Order Priority             Order_Date_c  \
Order Date
2011-01-01  106.140         35.46         Medium 2011-01-01 00:00:00+00:00
2011-01-02   92.880         53.08       Critical 2011-01-02 00:00:00+00:00
2011-01-03  -28.713          0.79         Medium 2011-01-03 00:00:00+00:00
2011-01-06    5.880          1.56         Medium 2011-01-06 00:00:00+00:00
2011-01-06    4.320          0.74         Medium 2011-01-06 00:00:00+00:00
...             ...           ...            ...                       ...
2013-10-31   -3.615          0.21           High 2013-10-31 00:00:00+00:00
2011-12-31    9.840          4.06           High 2011-12-31 00:00:00+00:00
2013-12-31  322.320         91.00         Medium 2013-12-31 00:00:00+00:00
2013-12-31   28.620         10.32            Low 2013-12-31 00:00:00+00:00
2013-12-31    6.060          1.35            Low 2013-12-31 00:00:00+00:00

            Year  Month  Day
Order Date
```

```
for index in cohort:
    weekly_cohort=cohort[index].resample('W').mean()
    x_week=weekly_cohort['Sales']
```

```
fig= plt.figure(figsize=(12,8))
ax = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(x_week, lags=50, zero=False, ax=ax)
plt.title('Autogression Plot for Cohort is :'+ str(Market_List[index]))
```

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

```
imp_mean
```

```
SimpleImputer()
```

```
array=np.zeros([50,6])
for index in cohort:
    monthly_cohort=cohort[index].resample('M').mean()
    x_Month=monthly_cohort['Sales']
    print(x_Month[0:3])
    fig= plt.figure(figsize=(12,8))
    ax = fig.add_subplot(211)
    fig = sm.graphics.tsa.plot_acf(x_Month, lags=40, zero=False, ax=ax)
    plt.title('Autogression Plot for Cohort is :'+ str(Market_List[index]))
```

```
training_data.head(3)
```

```
df3 = training_data.groupby(Vector2)['Sales'].mean()
```

```
df3
```

```
Market  Category
APAC    Furniture          559.682778
        Office Supplies    141.649615
        Technology         568.283771
Africa  Furniture          321.186781
        Office Supplies     88.914716
        Technology         349.462972
Canada  Furniture          185.928333
        Office Supplies    115.064725
        Technology         367.580556
EMEA    Furniture          299.088766
        Office Supplies     82.627403
        Technology         305.174846
EU      Furniture          522.323602
        Office Supplies    158.633852
        Technology         577.696658
LATAM   Furniture          331.942551
        Office Supplies     96.277158
        Technology         380.339615
US      Furniture          357.774597
        Office Supplies    115.823716
        Technology         463.420324
Name: Sales, dtype: float64
```

```
object2 = training_data.groupby(Vector2)['Sales']
```

```
object2.groups.keys()
```

```
dict_keys([('APAC', 'Furniture'), ('APAC', 'Office Supplies'), ('APAC', 'Technology'), (
```

```
validation_data.head(3)
```

```
validation_data_furniture=validation_data[validation_data['Category']=='Furniture']
validation_data_technology=validation_data[validation_data['Category']=='Technology']
validation_data_office=validation_data[validation_data['Category']=='Office Supplies']
```

```
#validation_data2=validation_data.resample('M').mean()
object_valid=validation_data.groupby(Vector2)['Sales']
```

```
object_valid
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7f9720dfb8d0>
```

```
#validation_data2
```

```
object_valid.groups.keys()
```

```
dict_keys([('APAC', 'Furniture'), ('APAC', 'Office Supplies'), ('APAC', 'Technology'), (
```

```
object2.dtype
```

```
Market  Category
APAC    Furniture          float64
        Office Supplies    float64
        Technology         float64
Africa  Furniture          float64
        Office Supplies    float64
        Technology         float64
Canada  Furniture          float64
        Office Supplies    float64
        Technology         float64
EMEA    Furniture          float64
        Office Supplies    float64
        Technology         float64
EU      Furniture          float64
        Office Supplies    float64
        Technology         float64
LATAM   Furniture          float64
```

```
        Office Supplies    float64
        Technology         float64
US      Furniture          float64
        Office Supplies    float64
        Technology         float64
Name: Sales, dtype: object
```

```
type(object2)
```

```
pandas.core.groupby.generic.SeriesGroupBy
```

```
dict1=dict(tuple(object2))
dict1
```

```
{('APAC', 'Furniture'): Order Date
 2011-01-01     113.6700
 2011-01-03     351.7500
 2011-01-03     238.9290
 2011-01-04     567.6000
 2011-01-06    1037.7096
                 ...
 2013-12-31     706.1580
 2013-12-31     499.4400
 2013-12-31     179.9658
 2013-12-31     143.3700
 2013-12-31     171.3600
 Name: Sales, Length: 1947, dtype: float64,
 ('APAC', 'Office Supplies'): Order Date
 2011-01-01    120.366
 2011-01-01     55.242
 2011-01-02    162.720
 2011-01-02    352.350
 2011-01-02     40.680
                ...
 2012-12-31      7.314
 2013-12-31    498.000
 2013-12-31    170.964
 2013-12-31     59.400
 2013-12-31     46.530
 Name: Sales, Length: 4951, dtype: float64,
 ('APAC', 'Technology'): Order Date
 2011-01-02     285.7800
 2011-01-03     214.7580
 2011-01-03      91.8720
 2011-01-08    1157.5800
 2011-01-08     516.9600
                 ...
 2013-10-31     220.6500
 2012-12-31     241.3200
 2013-12-31     671.9265
 2013-12-31     115.4400
```

```
2013-12-31      85.9320
Name: Sales, Length: 1958, dtype: float64,
('Africa', 'Furniture'): Order Date
2011-01-07     29.268
2011-01-08    922.860
2011-02-06     48.366
2011-02-08    275.400
2011-02-08    142.623
               ...
2014-03-31    347.880
2011-05-31     85.020
2012-08-31     17.550
2011-10-31    993.060
2011-12-31    506.040
Name: Sales, Length: 507, dtype: float64,
('Africa', 'Office Supplies'): Order Date
2011-01-01    408.300
2011-01-02    206.400
2011-01-03     25.317
2011-01-06     21.180
2011-01-06     10.860
```

```
List1=dict1.keys()
```

```
#pd.DataFrame.from_dict(dict1,orient='index')
```

```
for key, item in object2:
    print("Key is: " + str(key))
    print(str(item), "\n\n")
```

```
Key is: ('APAC', 'Furniture')
Order Date
2011-01-01     113.6700
2011-01-03     351.7500
2011-01-03     238.9290
2011-01-04     567.6000
2011-01-06    1037.7096
                 ...
2013-12-31     706.1580
2013-12-31     499.4400
2013-12-31     179.9658
2013-12-31     143.3700
2013-12-31     171.3600
Name: Sales, Length: 1947, dtype: float64


Key is: ('APAC', 'Office Supplies')
Order Date
2011-01-01    120.366
2011-01-01     55.242
```

```
2011-01-02    162.720
2011-01-02    352.350
2011-01-02     40.680
                ...
2012-12-31      7.314
2013-12-31    498.000
2013-12-31    170.964
2013-12-31     59.400
2013-12-31     46.530
Name: Sales, Length: 4951, dtype: float64


Key is: ('APAC', 'Technology')
Order Date
2011-01-02     285.7800
2011-01-03     214.7580
2011-01-03      91.8720
2011-01-08    1157.5800
2011-01-08     516.9600
                 ...
2013-10-31     220.6500
2012-12-31     241.3200
2013-12-31     671.9265
2013-12-31     115.4400
2013-12-31      85.9320
Name: Sales, Length: 1958, dtype: float64


Key is: ('Africa', 'Furniture')
Order Date
2011-01-07     29.268
2011-01-08    922.860
2011-02-06     48.366
2011-02-08    275.400
2011-02-08    142.623
                ...
2014-03-31    347.880
2011-05-31     85.020
```

```
for key, item in object_valid:
    print("Key is: " + str(key))
    print(str(item), "\n\n")
```

```
Key is: ('APAC', 'Furniture')
Order Date
2014-07-02     706.1580
2014-07-02     111.0600
2014-07-03     263.6550
2014-07-03     127.4130
2014-07-04     200.7600
                 ...
2014-10-31     106.0200
2014-12-31    1091.2806
2014-12-31    1048.7313
```

```
2014-12-31     292.7592
2014-12-31     364.5900
Name: Sales, Length: 482, dtype: float64


Key is: ('APAC', 'Office Supplies')
Order Date
2014-07-01    3810.9960
2014-07-01    1788.5880
2014-07-01      23.7600
2014-07-02     182.5500
2014-07-02      88.7400
                ...
2014-12-31      20.9244
2014-12-31      67.2000
2014-12-31      72.0000
2014-12-31      39.4200
2014-12-31      79.4700
Name: Sales, Length: 1226, dtype: float64


Key is: ('APAC', 'Technology')
Order Date
2014-07-01     116.1675
2014-07-02     340.4826
2014-07-03      61.8000
2014-07-03      81.4800
2014-07-04    3271.2000
                ...
2014-12-31     300.2400
2014-12-31     276.6000
2014-12-31     171.9900
2014-12-31     293.6208
2014-12-31      61.9740
Name: Sales, Length: 438, dtype: float64


Key is: ('Africa', 'Furniture')
Order Date
2014-07-03     293.220
2014-07-11     195.720
2014-08-05     383.220
2014-08-05     106.080
2014-08-05      50.370
                ...
2014-10-29     180.120
```

```
Market_List
```

```
array(['Africa', 'APAC', 'EMEA', 'EU', 'US', 'LATAM', 'Canada'],
      dtype=object)
```

```
Category_List=training_data2['Category'].unique()
```

```
object2.get_group(('APAC','Technology'))
```

```
Order Date
2011-01-02     285.7800
2011-01-03     214.7580
2011-01-03      91.8720
2011-01-08    1157.5800
2011-01-08     516.9600
                 ...
2013-10-31     220.6500
2012-12-31     241.3200
2013-12-31     671.9265
2013-12-31     115.4400
2013-12-31      85.9320
Name: Sales, Length: 1958, dtype: float64
```

```
#Create a Dataframe by looping
df1=object2.get_group((Market_List[0],Category_List[0])).to_frame()
```

```
object2
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x7f9720cc4450>
```

```
Category_List
```

```
array(['Office Supplies', 'Furniture', 'Technology'], dtype=object)
```

```
for i in range(len(Market_List)):
  for j in range(len(Category_List)):
     df1=object2.get_group((Market_List[0],Category_List[0])).to_frame()
```

```
df1
```

```
df1=df1.rename(columns={'Sales':'Sales-Geo1'})
```

```
df2=object2.get_group((Market_List[1],Category_List[0])).to_frame()
df2=df2.rename(columns={'Sales':'Sales-Geo2'}).copy()
```

```
df3=object2.get_group((Market_List[1],Category_List[0])).to_frame()
df3=df3.rename(columns={'Sales':'Sales-Geo3'})
```

```
result = df1.join(df2, how='outer').join(df3, how='outer')
result
```

```
mat=result.corr()
sns.clustermap(mat)
```

```
ax=result.iloc[:,:3].plot.area(fontsize=12)
ax.set_xlabel('Order Date')
plt.show()
```

Now creating dataframes for each cohort on the go and running model to predict sales

```
Market_List
Category_List
```

```
array(['Office Supplies', 'Furniture', 'Technology'], dtype=object)
```

```
#df=object2.get_group((Market_List[i],Category_List[j])).to_frame()
#df
#df2=df.resample('M').mean()
#df2['Sales'].isna().sum()
```

```
valid=object_valid.get_group((Market_List[0],Category_List[0])).to_frame()
```

```
valid
```

```
df_valid2=valid.resample('M').mean()
```

```
df_valid2.values
```

```
array([[144.0232    ],
       [ 87.13904545],
       [ 75.50384   ],
       [117.95869737],
       [ 40.77922222],
       [ 85.21735862],
       [ 77.55311628]])
```

```
#model=ARIMA(x_month,order=(1,1,1))
#model_fit=model.fit()
#output=model_fit.forecast(steps=7)[0]
```

```
output
```

```
(array([1808.02994444, 1811.64688889, 1815.26383333, 1818.88077778,
        1822.49772222, 1826.11466667]),
 array([3991.34238286, 5644.61052991, 6913.20779751, 7982.68476572,
        8924.91288955, 9776.75222675]),
 array([[ -6014.85737593,   9630.91726481],
        [ -9251.58645649,  12874.88023427],
        [-11734.37446743,  15364.9021341 ],
        [-13826.89386296,  17464.65541852],
        [-15670.01010645,  19315.00555089],
        [-17335.96758353,  20988.19691686]]))
```

```
predictions={}
error_list=[]
P=defaultdict()
error_dict=defaultdict()
k=0
for i in  range(len(Market_List)):
   for j  in range(len(Category_List)):
      df=object2.get_group((Market_List[i],Category_List[j])).to_frame()
      df2=df.resample('M').mean()
      nan_length=df2['Sales'].isna().sum()
      if nan_length >0 :
        print("Series in unforecastable for "+ str(Market_List[i])+str(Category_List[j]))
      else:
       x_month=df2['Sales']
       if(Category_List[j]=='Furniture'):
           model=ARIMA(x_month,order=(1,1,1))
           model_fit=model.fit()
```

```
        output=model_fit.forecast(steps=7)[0]
         #Create a validation set
        df2=object_valid.get_group((Market_List[i],Category_List[j])).to_frame()
        df_valid2=df2.resample('M').mean()#resampled mean
        valid_array=df_valid2.values
        valid_array=valid_array.reshape(-1,)
        q=MAPE(output,valid_array)
        error_list.append(q)
        key=str(Market_List[i])+ ':' +str(Category_List[j])
        if key not in predictions:
          predictions[key] = []
        if key not in error_dict:
          error_dict[key] = []
        predictions[key].append(output[0:])
        error_dict[key].append(q)
    else:
        model=ARIMA(x_month,order=(1,1,0))# Lowest Output from Validation Data
        model_fit=model.fit()
        output=model_fit.forecast(steps=7)[0]
         #Create a validation set
        df2=object_valid.get_group((Market_List[i],Category_List[j])).to_frame()
        df_valid2=df2.resample('M').mean()#resampled mean
        valid_array=df_valid2.values
        valid_array=valid_array.reshape(-1,)
        q=MAPE(output,valid_array)
        error_list.append(q)
        key=str(Market_List[i])+ ':' +str(Category_List[j])
        if key not in predictions:
          predictions[key] = []
        if key not in error_dict:
          error_dict[key]=[]
        predictions[key].append(output[0:])
        error_dict[key].append(q)
```

```
Series in unforecastable for CanadaFurniture
Series in unforecastable for CanadaTechnology
```

```
predictions
```

```
{'APAC:Furniture': [array([577.00310274, 567.0248447 , 565.18417614, 565.16215599,
         565.54658077, 566.02184086, 566.51740149])],
 'APAC:Office Supplies': [array([126.4633248 , 115.37885947, 119.6071035 , 117.44223753,
         117.94651746, 117.33641959, 117.19157837])],
 'APAC:Technology': [array([568.19832424, 592.53390261, 582.69244287, 587.62154665,
         586.16713802, 587.47154293, 587.58364936])],
```

```
'Africa:Furniture': [array([382.71772087, 353.7846061 , 362.48615728, 362.95814467,
       365.22968889, 367.10772443, 369.07180843])],
'Africa:Office Supplies': [array([138.37870045, 129.9897935 , 136.15542663, 133.819640
       136.44959914, 136.179028  , 137.60267858])],
'Africa:Technology': [array([340.56636179, 313.11592427, 323.53424581, 320.57435136,
       322.34069214, 322.43735593, 323.1238806 ])],
'Canada:Office Supplies': [array([23.93210362, 22.70375237, 19.94537183, 18.0847666 , 1
       13.61908406, 11.35942191])],
'EMEA:Furniture': [array([270.19426342, 266.35693163, 264.11627356, 262.27490029,
       260.53337732, 258.8168242 , 257.10651537])],
'EMEA:Office Supplies': [array([72.38438875, 68.38276532, 69.74330457, 67.63113846, 67
       66.44832809, 66.07195477])],
'EMEA:Technology': [array([354.56843639, 369.94225994, 361.8142971 , 360.5516579 ,
       357.28352568, 354.60123652, 351.74781134])],
'EU:Furniture': [array([499.47584578, 505.70438276, 504.47205809, 503.65719596,
       502.81897527, 501.98206157, 501.14507475])],
'EU:Office Supplies': [array([158.42308938, 151.78115033, 154.52564045, 153.75028726,
       154.29484995, 154.34445369, 154.57966348])],
'EU:Technology': [array([551.33969716, 558.0973672 , 557.2757378 , 560.14066053,
       561.2124536 , 563.15642042, 564.67616413])],
'LATAM:Furniture': [array([311.28384028, 310.78896568, 309.59940029, 308.45902742,
       307.31517111, 306.17156148, 305.02793438])],
'LATAM:Office Supplies': [array([94.17700083, 90.08940364, 92.10962319, 91.89531158, 92
       92.80260493, 93.21612938])],
'LATAM:Technology': [array([391.27992737, 378.06426782, 388.16560371, 387.60691901,
       391.92176269, 394.00853573, 397.11393396])],
'US:Furniture': [array([327.24259119, 334.46110396, 334.99179224, 334.26873571,
       333.31064371, 332.30849037, 331.29807699])],
'US:Office Supplies': [array([79.04816066, 78.04465315, 77.64990508, 76.99330361, 76.4
       75.85692048, 75.28534441])],
'US:Technology': [array([443.20411684, 455.57937395, 451.57399703, 453.80355394,
       453.65991823, 454.41958692, 454.83543235])]}
```

```
predictions['Canada:Office Supplies'][0]
```

```
array([23.93210362, 22.70375237, 19.94537183, 18.0847666 , 15.69737374,
       13.61908406, 11.35942191])
```

```
result=defaultdict()
for key, value in predictions.items():
       result[key]=value[0]
```

```
FPRECAST=pd.DataFrame(result)
error_cast=pd.DataFrame(error_dict)
```

```
FORECAST=FPRECAST.T
errorcast=error_cast.T
```

```
FORECAST=FORECAST.reset_index().copy()
```

```
errorcast=errorcast.reset_index().copy()
```

```
FORECAST[['Region','Category']]=FORECAST['index'].str.split(':',expand=True)
errorcast[['Region','Category']]=errorcast['index'].str.split(':',expand=True)
errorcast
del errorcast['index']
del FORECAST['index']
final_cast=errorcast.merge(FORECAST,on=['Region','Category'],how='left')
```

```
final_cast=errorcast.merge(FORECAST,on=['Region','Category'],how='left')
```

```
from google.colab import files
import pandas as pd
final_cast.to_csv('forecast_file.csv')
files.download('forecast_file.csv')
```

```
# Validation Set Run for Results2
# predictions4={}
# error_list4=[]
# for i in  range(len(Market_List)):
#    for j  in range(len(Category_List)):
#       df=object2.get_group((Market_List[i],Category_List[j])).to_frame()
#       df2=df.resample('M').mean()
#       nan_length=df2['Sales'].isna().sum()
#       if nan_length >0 :
#         print("Series in unforecastable for "+ str(Market_List[i])+str(Category_List[j]))
#       else:
#        x_month=df2['Sales']
#        model=ARIMA(x_month,order=results2.order)
#        model_fit=model.fit()
#        output=model_fit.forecast(steps=7)[0]
#        #Create a validation set
#        df2=object_valid.get_group((Market_List[i],Category_List[j])).to_frame()
#        df_valid2=df2.resample('M').mean()#resampled mean
#        valid_array=df_valid2.values
#        valid_array=valid_array.reshape(-1,)
#        q=MAPE(output,valid_array)
#        error_list4.append(q)
#        key=str(Market_List[i])+ ':' +str(Category_List[j])
#        if key not in predictions4:
#            predictions4[key] = []
#        predictions4[key].append(output.values)
```

```
r=pd.DataFrame(predictions.items())
```

```
r.head(1)
```

```
# predictions={}
# error_list=[]
# for i in  range(len(Market_List)):
  #  for j  in range(len(Category_List)):
     # df=object2.get_group((Market_List[i],Category_List[j])).to_frame()
      # # df2=df.resample('M').mean()
      # nan_length=df2['Sales'].isna().sum()
      # if nan_length >0 :
       # print("Series in unforecastable for "+ str(Market_List[i])+str(Category_List[j]))
      # # # # else:
      # # # # #  model=SARIMAX(df2,order=(0,1,1),trend='c')
      # # # # #  model_fit=model.fit()
      # # # # #  output=model_fit.forecast(steps=7)
       #Create a validation set
      # # # # #  df_valid=object_valid.get_group((Market_List[i],Category_List[j])).to_frame(
      # # # # #  df_valid2=df_valid.resample('M').mean()#resampled mean
      # # # # #  valid_array=df_valid2.values#values
      # # # # #  valid_array_new=single_array(valid_array)
      # # # # #  q=MAPE(output,valid_array)
      # # # # #  error_list.append(q)
      # # # #  key=str(Market_List[i])+ str(Category_List[j])
      # # #  if key not in predictions:
          # # #  predictions[key] = []
      #  predictions[key].append(output)
```

```
#single_array(valid_array)
```

```
predictions
```

```
{'APAC:Furniture': [array([577.00310274, 567.0248447 , 565.18417614, 565.16215599,
         565.54658077, 566.02184086, 566.51740149])],
 'APAC:Office Supplies': [array([126.4633248 , 115.37885947, 119.6071035 , 117.44223753,
         117.94651746, 117.33641959, 117.19157837])],
 'APAC:Technology': [array([568.19832424, 592.53390261, 582.69244287, 587.62154665,
         586.16713802, 587.47154293, 587.58364936])],
 'Africa:Furniture': [array([382.71772087, 353.7846061 , 362.48615728, 362.95814467,
         365.22968889, 367.10772443, 369.07180843])],
```

```
 'Africa:Office Supplies': [array([138.37870045, 129.9897935 , 136.15542663, 133.819640
        136.44959914, 136.179028  , 137.60267858])],
 'Africa:Technology': [array([340.56636179, 313.11592427, 323.53424581, 320.57435136,
        322.34069214, 322.43735593, 323.1238806 ])],
 'Canada:Office Supplies': [array([23.93210362, 22.70375237, 19.94537183, 18.0847666 , 
        13.61908406, 11.35942191])],
 'EMEA:Furniture': [array([270.19426342, 266.35693163, 264.11627356, 262.27490029,
        260.53337732, 258.8168242 , 257.10651537])],
 'EMEA:Office Supplies': [array([72.38438875, 68.38276532, 69.74330457, 67.63113846, 67
        66.44832809, 66.07195477])],
 'EMEA:Technology': [array([354.56843639, 369.94225994, 361.8142971 , 360.5516579 ,
        357.28352568, 354.60123652, 351.74781134])],
 'EU:Furniture': [array([499.47584578, 505.70438276, 504.47205809, 503.65719596,
        502.81897527, 501.98206157, 501.14507475])],
 'EU:Office Supplies': [array([158.42308938, 151.78115033, 154.52564045, 153.75028726,
        154.29484995, 154.34445369, 154.57966348])],
 'EU:Technology': [array([551.33969716, 558.0973672 , 557.2757378 , 560.14066053,
        561.2124536 , 563.15642042, 564.67616413])],
 'LATAM:Furniture': [array([311.28384028, 310.78896568, 309.59940029, 308.45902742,
        307.31517111, 306.17156148, 305.02793438])],
 'LATAM:Office Supplies': [array([94.17700083, 90.08940364, 92.10962319, 91.89531158, 9
        92.80260493, 93.21612938])],
 'LATAM:Technology': [array([391.27992737, 378.06426782, 388.16560371, 387.60691901,
        391.92176269, 394.00853573, 397.11393396])],
 'US:Furniture': [array([327.24259119, 334.46110396, 334.99179224, 334.26873571,
        333.31064371, 332.30849037, 331.29807699])],
 'US:Office Supplies': [array([79.04816066, 78.04465315, 77.64990508, 76.99330361, 76.4
        75.85692048, 75.28534441])],
 'US:Technology': [array([443.20411684, 455.57937395, 451.57399703, 453.80355394,
        453.65991823, 454.41958692, 454.83543235])]}
```

```
p=predictions.keys()
```

```
error_list
```

```
[74.10107695301738,
 51.23174164385349,
 20.500540483721092,
 35.40469595506458,
 12.466914208168568,
 9.613829398686233,
 18.823622185382515,
 59.65684759435076,
 34.471952695533034,
 11.25929400417819,
 13.457424583810754,
 10.597945608323489,
```

```
40.34440987092168,
12.85088803596012,
20.612513132820588,
14.411178613143107,
16.431059703288888,
9.79532657666737,
76.22074271531096]
```

```
g=sns.histplot(error_list).set(title=" Histogram of MAPE error for different time series")
```

```
for k,v in predictions.items():
  m=v[0]
  fig= plt.figure(figsize=(6,6))
  x=np.linspace(1,7,7)
  ax = fig.add_subplot(211)
  ax.set_xticks(x)
  ax.set_xticklabels(x.astype(int))
  ax.plot(x,m)
  ax.set_title(str(k))
```

```
import re
```

```
for k,v  in predictions.items():
   k=str(k)
   m=k.split(':')
   print(m[0]+':'+m[1])
```

```
Africa:Office Supplies
Africa:Furniture
Africa:Technology
APAC:Office Supplies
APAC:Furniture
APAC:Technology
EMEA:Office Supplies
EMEA:Furniture
EMEA:Technology
EU:Office Supplies
EU:Furniture
EU:Technology
US:Office Supplies
US:Furniture
US:Technology
LATAM:Office Supplies
LATAM:Furniture
LATAM:Technology
Canada:Office Supplies
```

```
for k,v in predictions.items():
   key=str(k).split(':')
   m=v[0]
   fig= plt.figure(figsize=(6,6))
   ax = fig.add_subplot(211)
   ax.plot(df_valid2.index,m)
   ax.set_title('Prediction on Validation Set  :'+ key[0]+'\n'+key[1])
```

```
##Lets create clustering of data
```

RECURRENT NEURAL NET CREATING A SEQUENCE OF 4 TIME SERIES LENGTH

```
# Step 1 Libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

```
import numpy as np
```

1 Furniture Product

```
x_train_furniture=df_first.values
```

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

```
SEQ_LEN = 3
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)## Each X t

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'])
```

```
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 = 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')
```

Double-click (or enter) to edit

```
x_test=df_valid_furniture.values
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_valid = scaler.fit_transform(x_test)
```

```
X_test=scaled_valid[0:-SEQ_LEN-1,:].reshape(-1,1,1)#NEW SHAPE BY TAKING VECTOR OF 4
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)
```

Double-click (or enter) to edit

y_hat_valid=model.predict(X_test)

```
df_valid_furniture
```

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

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<
```

```
y_hat_valid
```

```
array([[0.3568446 ],
       [0.48241955],
       [0.5000204 ]], dtype=float32)
```

```
y_hat_valid_unscaled=scaler.inverse_transform(y_hat_valid)
```

```
yactual=x_test[SEQ_LEN:-1,-1]
```

```
mape_furniture_valid=MAPE(y_hat_valid,yactual)
```

```
list_m=[mape_furniture_model1,mape_furniture_model2,mape_furniture_model3,mape_furniture_vali
x=['model-110 ARIMA ','model: 100 ARIMA','model:3,2,0 SARIMAX','Model-RNN,Lag=3,50_LSTM']
```

```
sns.barplot(list_m,x).set_title("Validation Error MAPE-Furniture")
```

italicized text

```
Data=pd.read_csv('/content/drive/My Drive/Wisconsin_Project/inflation.csv')
```

```
Data.head(1)
```

```
Data['LOCATION'].value_counts()
```

```
CHE          195
IRL          195
EST          195
TUR          195
BEL          195
IDN          195
SVN          195
NLD          195
MEX          195
LUX          195
CHL          195
KOR          195
ISL          195
HUN          195
COL          195
LTU          195
OECDE        194
OECD         194
G-7          194
ZAF          194
LVA          194
RUS          194
ISR          194
SAU          194
IND          194
EA19         194
G-20         194
CHN          194
AUT          194
BRA          194
USA          194
CAN          194
CZE          194
DNK          194
FIN          194
FRA          194
DEU          194
GRC          194
ITA          194
JPN          194
NOR          194
POL          194
PRT          194
SVK          194
ESP          194
SWE          194
GBR          194
EU27_2020    194
ARG           51
Name: LOCATION, dtype: int64
```

```
Data2=Data[Data['LOCATION']=='USA']
```

```
Data3=Data2[['TIME','Value']]
```

```
Data3['TIME']=pd.to_datetime(Data3['TIME'])
```

```
/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.
```

```
Data3['Year']=Data3['TIME'].dt.strftime("%Y")# Only Order Year
Data3['Month']=Data3['TIME'].dt.strftime("%m")
```

```
/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.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: 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_
```

```
Data3=Data3.rename(columns={'TIME':'Synthetic_Date'})
```

```
# Data3=Data3.rename(columns={'Order Date':'Synthetic Date'}).copy()
# Data3=Data3.rename(columns={'Synthetic Date':'Synthetic_Date'})
```

```
Data4=Data3.reset_index()
```

```
Data5=Data4.iloc[:,1:]
```

```
Data5
```

```
df_first_oecd=df_first.reset_index()
```

```
df_first_oecd.head(2)
```

```
df_first_oecd.dtypes
```

```
Synthetic_Date      datetime64[ns]
Monthly_Quantity           float64
dtype: object
```

```
Data4.head(1)
```

```
Data5.tail(2)
```

```
oecd_data_furniture_us=df_first_oecd.merge(Data5,on='Synthetic_Date',how='left')
```

```
oecd_data_furniture_us
```

```
oecd_data_furniture_us2=oecd_data_furniture_us.iloc[:,1:3]


x_train_furniture=oecd_data_furniture_us2.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(x_train_furniture)
SEQ_LEN = 3
DATA_LEN = scaled.shape[0]# Length

X_train = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,2)#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(-1,1,2), axis=1)## Each X t

Y_train = scaled[SEQ_LEN:-1,0]#
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 = 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')
```

```
def correction_nn(X):
    X.reset_index
```

```
df_valid_furn=df_valid_furniture.reset_index()
```

```
df_valid_furn.head(10)
```

```
x_valid2=df_valid_furn.merge(Data5,on='Synthetic_Date',how='left')
```

```
x_valid3=x_valid2.iloc[:,1:3]
```

```
x_valid3
```

```
#Converting validation set to a tensor/3 dimensitonal matrix
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(x_valid3)
```

```
x_valid = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,2)
```

```
x_valid.shape
```

```
(3, 1, 2)
```

```
x_valid = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,2)#preparing training data from 0:-6

for i in range(1,SEQ_LEN):
  x_valid = np.append(x_valid, scaled[i:-SEQ_LEN+i-1,:].reshape(-1,1,2), axis=1)## Each X tra
```

```
Y_valid=x_valid3.iloc[SEQ_LEN:-1,0]
```

```
Y_valid_val=Y_valid.values
Y_valid_n = Y_valid_val.reshape(1, -1)#  converts into two dimensional array
```

```
Y_valid_n.shape
```

```
(1, 3)
```

```
scaler_1=MinMaxScaler(feature_range=(0,1))
y_valid=scaler_1.fit_transform(Y_valid_n)
```

```
x_valid.shape
```

```
(3, 3, 2)
```

```
y_hat=model.predict(x_valid)
```

```
y_hat_new=y_hat.reshape(1,-1)
```

```
y_hat_new.shape
```

```
(1, 3)
```

```
y_hat_unscale_new=scaler_1.inverse_transform(y_hat_new)
```

```
y_hat_unscale_new
```

```
array([[9531.891 , 5955.3486, 8364.194 ]], dtype=float32)
```

```
Y_valid_val
```

```
array([9531.501, 5954.845, 8363.685])
```

```
MAPE(y_hat_unscale_new,Y_valid_val)
```

```
0.006211714056618732
```

```
def rnn_training_model(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(50, 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
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_valid = scaler.fit_transform(x_test)
    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)
    return mape1
```

Data Preperation for merging data with OECD DATA FOR 3 PRODUCTS

```
oecd_data_furniture_us=df_first_oecd.merge(Data5,on='Synthetic_Date',how='left')
```

```
oecd_data_tech_us=df_second_oecd.merge(Data5,on='Synthetic_Date',how='left')
```

```
oecd_data_office_us=df_third_oecd.merge(Data5,on='Synthetic_Date',how='left')
oecd_data_office_us2=oecd_data_office_us.iloc[:,1:3]
oecd_data_tech_us2=oecd_data_tech_us.iloc[:,1:3]
```

```
oecd_data_furniture_us2=oecd_data_furniture_us.iloc[:,1:3]
```

```
x_valid3_tech
```

```
def rnn_multivariate(oecd_product,oecd_valid,k):
    x_train=oecd_product.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(-1,1,2)#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(-1,1,2), axis=1)## Each
    Y_train = scaled[SEQ_LEN:-1,0]#
    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 = 100
    validation_split = 0.1
    history = model.fit(X_train, Y_train, batch_size=128,epochs=epochs,validation_split=valid
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(x_valid3)
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled = scaler.fit_transform(x_valid3)
    x_valid = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,2)
    x_valid = scaled[0:-SEQ_LEN-1,:].reshape(-1,1,2)#preparing training data from 0:-6
    for i in range(1,SEQ_LEN):
      x_valid = np.append(x_valid, scaled[i:-SEQ_LEN+i-1,:].reshape(-1,1,2), axis=1)#
```

```
    Y_valid=oecd_valid.iloc[SEQ_LEN:-1,0]
    Y_valid_val=Y_valid.values
    Y_valid_n = Y_valid_val.reshape(1, -1)
    scaler_1=MinMaxScaler(feature_range=(0,1))
    y_valid=scaler_1.fit_transform(Y_valid_n)
    y_hat=model.predict(x_valid)
    y_hat_new=y_hat.reshape(1,-1)
    y_hat_unscale_new=scaler_1.inverse_transform(y_hat_new)
    score_product=MAPE(y_hat_unscale_new,Y_valid_val)
    return score_product

p_tech=rnn_multivariate(oecd_data_tech_us2,x_valid3_tech,3)
```

```
Epoch 1/100
2/2 [==============================] - 2s 379ms/step - loss: 0.0384 - mae: 0.1349 -
Epoch 2/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0330 - mae: 0.1151 - v
Epoch 3/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0289 - mae: 0.1027 - v
Epoch 4/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0260 - mae: 0.0969 - v
Epoch 5/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0242 - mae: 0.0945 - v
Epoch 6/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0233 - mae: 0.0968 - v
Epoch 7/100
2/2 [==============================] - 0s 25ms/step - loss: 0.0231 - mae: 0.1005 - v
Epoch 8/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0232 - mae: 0.1034 - v
Epoch 9/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0234 - mae: 0.1063 - v
Epoch 10/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0237 - mae: 0.1086 - v
Epoch 11/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0235 - mae: 0.1092 - v
Epoch 12/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0232 - mae: 0.1082 - v
Epoch 13/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0227 - mae: 0.1067 - v
Epoch 14/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0222 - mae: 0.1047 - v
Epoch 15/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0217 - mae: 0.1023 - v
Epoch 16/100
2/2 [==============================] - 0s 24ms/step - loss: 0.0213 - mae: 0.0997 - v
Epoch 17/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0210 - mae: 0.0969 - v
Epoch 18/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0208 - mae: 0.0945 - v
Epoch 19/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0207 - mae: 0.0928 - v
Epoch 20/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0207 - mae: 0.0920 - v
Epoch 21/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0207 - mae: 0.0915 - v
```

```
Epoch 22/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0207 - mae: 0.0913 - v
Epoch 23/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0206 - mae: 0.0913 - v
Epoch 24/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0205 - mae: 0.0914 - v
Epoch 25/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0204 - mae: 0.0918 - v
Epoch 26/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0203 - mae: 0.0924 - v
Epoch 27/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0202 - mae: 0.0932 - v
Epoch 28/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0202 - mae: 0.0937 - v
Epoch 29/100
```

```
p_office=rnn_multivariate(oecd_data_office_us2,x_valid3_off,3)
```

```
Epoch 1/100
2/2 [==============================] - 2s 380ms/step - loss: 0.1092 - mae: 0.2693 -
Epoch 2/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0961 - mae: 0.2440 - v
Epoch 3/100
2/2 [==============================] - 0s 23ms/step - loss: 0.0846 - mae: 0.2210 - v
Epoch 4/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0743 - mae: 0.2002 - v
Epoch 5/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0656 - mae: 0.1836 - v
Epoch 6/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0581 - mae: 0.1702 - v
Epoch 7/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0520 - mae: 0.1594 - v
Epoch 8/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0473 - mae: 0.1527 - v
Epoch 9/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0435 - mae: 0.1482 - v
Epoch 10/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0412 - mae: 0.1471 - v
Epoch 11/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0398 - mae: 0.1470 - v
Epoch 12/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0391 - mae: 0.1484 - v
Epoch 13/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0392 - mae: 0.1506 - v
Epoch 14/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0395 - mae: 0.1527 - v
Epoch 15/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0402 - mae: 0.1555 - v
Epoch 16/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0405 - mae: 0.1572 - v
Epoch 17/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0405 - mae: 0.1577 - v
Epoch 18/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0400 - mae: 0.1566 - v
Epoch 19/100
```

```
2/2 [==============================] - 0s 21ms/step - loss: 0.0391 - mae: 0.1541 - v
Epoch 20/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0382 - mae: 0.1513 - v
Epoch 21/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0375 - mae: 0.1483 - v
Epoch 22/100
2/2 [==============================] - 0s 27ms/step - loss: 0.0369 - mae: 0.1455 - v
Epoch 23/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0365 - mae: 0.1438 - v
Epoch 24/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0362 - mae: 0.1423 - v
Epoch 25/100
2/2 [==============================] - 0s 22ms/step - loss: 0.0359 - mae: 0.1411 - v
Epoch 26/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0358 - mae: 0.1400 - v
Epoch 27/100
2/2 [==============================] - 0s 20ms/step - loss: 0.0357 - mae: 0.1390 - v
Epoch 28/100
2/2 [==============================] - 0s 21ms/step - loss: 0.0356 - mae: 0.1380 - v
Epoch 29/100
```

```
p_office
```

```
0.004488847502851159
```

```
p=rnn_multivariate(oecd_data_furniture_us2,x_valid3,3)
```

```
Epoch 1/100
1/1 [==============================] - 2s 2s/step - loss: 0.1892 - mae: 0.3682 - val_
Epoch 2/100
1/1 [==============================] - 0s 25ms/step - loss: 0.1786 - mae: 0.3542 - v
Epoch 3/100
1/1 [==============================] - 0s 24ms/step - loss: 0.1684 - mae: 0.3402 - v
Epoch 4/100
1/1 [==============================] - 0s 22ms/step - loss: 0.1586 - mae: 0.3265 - v
Epoch 5/100
1/1 [==============================] - 0s 23ms/step - loss: 0.1493 - mae: 0.3135 - v
Epoch 6/100
1/1 [==============================] - 0s 23ms/step - loss: 0.1404 - mae: 0.3005 - v
Epoch 7/100
1/1 [==============================] - 0s 23ms/step - loss: 0.1319 - mae: 0.2875 - v
Epoch 8/100
1/1 [==============================] - 0s 23ms/step - loss: 0.1239 - mae: 0.2747 - v
Epoch 9/100
1/1 [==============================] - 0s 24ms/step - loss: 0.1163 - mae: 0.2626 - v
Epoch 10/100
1/1 [==============================] - 0s 23ms/step - loss: 0.1092 - mae: 0.2516 - v
Epoch 11/100
1/1 [==============================] - 0s 24ms/step - loss: 0.1025 - mae: 0.2417 - v
Epoch 12/100
1/1 [==============================] - 0s 26ms/step - loss: 0.0963 - mae: 0.2323 - v
Epoch 13/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0905 - mae: 0.2232 - v
```

```
Epoch 14/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0852 - mae: 0.2147 - v
Epoch 15/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0804 - mae: 0.2077 - v
Epoch 16/100
1/1 [==============================] - 0s 24ms/step - loss: 0.0761 - mae: 0.2019 - v
Epoch 17/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0722 - mae: 0.1968 - v
Epoch 18/100
1/1 [==============================] - 0s 24ms/step - loss: 0.0689 - mae: 0.1933 - v
Epoch 19/100
1/1 [==============================] - 0s 22ms/step - loss: 0.0660 - mae: 0.1907 - v
Epoch 20/100
1/1 [==============================] - 0s 25ms/step - loss: 0.0637 - mae: 0.1880 - v
Epoch 21/100
1/1 [==============================] - 0s 25ms/step - loss: 0.0618 - mae: 0.1855 - v
Epoch 22/100
1/1 [==============================] - 0s 28ms/step - loss: 0.0604 - mae: 0.1837 - v
Epoch 23/100
1/1 [==============================] - 0s 25ms/step - loss: 0.0594 - mae: 0.1823 - v
Epoch 24/100
1/1 [==============================] - 0s 26ms/step - loss: 0.0588 - mae: 0.1819 - v
Epoch 25/100
1/1 [==============================] - 0s 26ms/step - loss: 0.0585 - mae: 0.1836 - v
Epoch 26/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0584 - mae: 0.1868 - v
Epoch 27/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0586 - mae: 0.1898 - v
Epoch 28/100
1/1 [==============================] - 0s 23ms/step - loss: 0.0589 - mae: 0.1927 - v
```

```
Product_List=['Office_Supply','Furniture','Technology']
error_score=[p_office,p,p_tech]
sns.barplot(Product_List,error_score).set_title("Multivariate RNN")
```