***PRODUCT DEMAND PREDICTION WITH***

***MACHINE LEARNING***

***INRTODUCTION :***

***IMPORT***

import os

for dirname,\_,filenames in os.walk('/kaggle/input'):

for filename in filenames:

print (os.path.join(dirname,filename))

import pandas as pd # Data handling and managing

import numpy as np # Handiling linear Algera

import seaborn as sn

import matplotlib.pyplot as plt

%matplotlib inline

df = pd.read\_csv('../input/productdemandforecasting/Historical Product Demand.csv', parse\_dates=['Date'])

df.head(100) # Getting the first 100 rows to view the records

#df.shape

Product\_Code Warehouse Product\_Category Date Order\_Demand

0 Product\_0993 Whse\_J Category\_028 2012-07-27 100

1 Product\_0979 Whse\_J Category\_028 2012-01-19 500

2 Product\_0979 Whse\_J Category\_028 2012-02-03 500

3 Product\_0979 Whse\_J Category\_028 2012-02-09 500

4 Product\_0979 Whse\_J Category\_028 2012-03-02 500

... ... ... ... ... ...

95 Product\_1512 Whse\_J Category\_019 2012-06-15 30000

96 Product\_1512 Whse\_J Category\_019 2012-06-20 30000

97 Product\_1512 Whse\_J Category\_019 2012-07-04 30000

98 Product\_1274 Whse\_J Category\_019 2012-06-19 1000

99 Product\_1451 Whse\_J Category\_019 2012-06-26 1000

df.dtypes

Product\_Code object

Warehouse object

Product\_Category object

Date datetime64[ns]

Order\_Demand object

dtype: object

# Check for the columns which got has the NaN values

print(df.isnull().any().sum(), ' / ', len(df.columns))

# Check any number of data points with NaN

print(df.isnull().any(axis=1).sum(),'/', len(df))

df.dropna(axis=0, inplace=True) #Remove all the rows with null values

df.reset\_index(drop=True)

df.sort\_values('Date')[1:50]

Product\_Code Warehouse Product\_Category Date Order\_Demand

72252 Product\_1724 Whse\_A Category\_003 2011-05-31 108

8431 Product\_1521 Whse\_S Category\_019 2011-06-24 85000

8432 Product\_1521 Whse\_S Category\_019 2011-06-24 7000

72669 Product\_1507 Whse\_C Category\_019 2011-09-02 1250

17249 Product\_0608 Whse\_C Category\_001 2011-09-27 5

17250 Product\_1933 Whse\_C Category\_001 2011-09-27 23

74615 Product\_0875 Whse\_C Category\_023 2011-09-30 5450

131426 Product\_0125 Whse\_S Category\_011 2011-10-20 (2)

131429 Product\_0412 Whse\_S Category\_007 2011-10-20 (2)

75193 Product\_0642 Whse\_C Category\_019 2011-10-31 3

121820 Product\_0202 Whse\_A Category\_007 2011-11-04 (100)

121819 Product\_0202 Whse\_A Category\_007 2011-11-04 (400)

131028 Product\_2143 Whse\_S Category\_009 2011-11-18 (25)

131031 Product\_0131 Whse\_S Category\_021 2011-11-18 (12)

131032 Product\_0288 Whse\_S Category\_021 2011-11-18 (50)

44450 Product\_0980 Whse\_A Category\_028 2011-11-18 4000

131027 Product\_2138 Whse\_S Category\_009 2011-11-18 (49)

131026 Product\_2137 Whse\_S Category\_009 2011-11-18 (25)

44795 Product\_0965 Whse\_A Category\_006 2011-11-18 1

44798 Product\_0965 Whse\_A Category\_006 2011-11-21 2

44797 Product\_0965 Whse\_A Category\_006 2011-11-21 5

44796 Product\_0965 Whse\_A Category\_006 2011-11-21 3

119561 Product\_0980 Whse\_A Category\_028 2011-11-21 100

107159 Product\_0138 Whse\_J Category\_007 2011-11-22 1852

107158 Product\_0138 Whse\_J Category\_007 2011-11-22 188

111727 Product\_0982 Whse\_A Category\_028 2011-11-22 3700

44102 Product\_0980 Whse\_A Category\_028 2011-11-23 1000

71915 Product\_0980 Whse\_A Category\_028 2011-11-23 200

44103 Product\_0980 Whse\_A Category\_028 2011-11-24 2000

73101 Product\_1264 Whse\_C Category\_019 2011-11-24 40000

45010 Product\_0982 Whse\_A Category\_028 2011-11-24 500

12858 Product\_1235 Whse\_C Category\_019 2011-11-25 16000

6702 Product\_1190 Whse\_C Category\_019 2011-11-25 1250

124567 Product\_0122 Whse\_A Category\_021 2011-11-28 310

72124 Product\_0987 Whse\_A Category\_028 2011-11-29 13100

45012 Product\_0982 Whse\_A Category\_028 2011-11-29 100

45011 Product\_0982 Whse\_A Category\_028 2011-11-29 250

72125 Product\_0987 Whse\_A Category\_028 2011-11-29 400

41105 Product\_0982 Whse\_A Category\_028 2011-11-29 500

72801 Product\_0208 Whse\_C Category\_007 2011-11-29 2

45233 Product\_0981 Whse\_A Category\_028 2011-11-30 400

113184 Product\_0981 Whse\_A Category\_028 2011-12-01 50

76479 Product\_1102 Whse\_S Category\_004 2011-12-01 1200

111728 Product\_0982 Whse\_A Category\_028 2011-12-01 4500

45013 Product\_0981 Whse\_A Category\_028 2011-12-02 2500

93055 Product\_0504 Whse\_J Category\_015 2011-12-05 1

107170 Product\_0138 Whse\_J Category\_007 2011-12-05 240

120622 Product\_0967 Whse\_A Category\_006 2011-12-05 10

124709 Product\_1095 Whse\_A Category\_024 2011-12-05 2

#df.dropna().sum()

df['Order\_Demand']=df['Order\_Demand'].str.replace('(',"")

df['Order\_Demand']=df['Order\_Demand'].str.replace(')',"")

df.head(100)

#Since the "()" has been removed , Now i Will change the data type.

df['Order\_Demand'] = df['Order\_Demand'].astype('int64')

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:1: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single character regular expressions will\*not\* be treated as literal strings when regex=True.

"""Entry point for launching an IPython kernel.

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:2: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single character regular expressions will\*not\* be treated as literal strings when regex=True.

df.sort\_values('Date')[10:20]

Product\_Code Warehouse Product\_Category Date Order\_Demand

75193 Product\_0642 Whse\_C Category\_019 2011-10-31 3

121820 Product\_0202 Whse\_A Category\_007 2011-11-04 100

121819 Product\_0202 Whse\_A Category\_007 2011-11-04 400

131028 Product\_2143 Whse\_S Category\_009 2011-11-18 25

131031 Product\_0131 Whse\_S Category\_021 2011-11-18 12

131032 Product\_0288 Whse\_S Category\_021 2011-11-18 50

44450 Product\_0980 Whse\_A Category\_028 2011-11-18 4000

131027 Product\_2138 Whse\_S Category\_009 2011-11-18 49

131026 Product\_2137 Whse\_S Category\_009 2011-11-18 25

44795 Product\_0965 Whse\_A Category\_006 2011-11-18 1

#Get the Hieghest and lowest dates in the dataset.

df['Date'].min() , df['Date'].max()

(Timestamp('2011-01-08 00:00:00'), Timestamp('2017-01-09 00:00:00'))

from scipy.stats import norm, skew #Import Norm and skew for some statistics

from scipy import stats #Import stats

import statsmodels.api as sm #for decomposing the trends, seasonality etc.

from statsmodels.tsa.statespace.sarimax import SARIMAX #for the Seasonal Forecast

#Lets check the ditribution of the target variable (Order\_Demand)

from matplotlib import rcParams

# figure size in inches

rcParams['figure.figsize'] = 10,5

sn.distplot(df['Order\_Demand'],fit=norm)

#Get the QQ-plot

fig = plt.figure()

res = stats.probplot(df['Order\_Demand'], plot=plt)

plt.show()

/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

warnings.warn(msg, FutureWarning)

df['Warehouse'].value\_counts().sort\_values(ascending=False)

Whse\_J 764447

Whse\_A 142335

Whse\_S 88200

Whse\_C 42354

Name: Warehouse, dtype: int64

#Now I will get the amount of orders shipped by each warehouse.

df.groupby('Warehouse').sum().sort\_values('Order\_Demand', ascending = False)

Order\_Demand

Warehouse

Whse\_J 3363200396

Whse\_S 1038024700

Whse\_C 585071404

Whse\_A 147877431

df['Date'] = pd.to\_datetime(df['Date'])

df['Year'] = df['Date'].dt.year

df2 = df[['Year', 'Warehouse', 'Order\_Demand']].groupby(['Year', 'Warehouse'], as\_index=False).count()

df2 = df2.pivot(index='Year', columns='Warehouse', values='Order\_Demand')

df2.index = df2.index.map(int) # let's change the index values of df2 to type integer for plotting

df2.plot(kind='area', stacked=False, figsize=(20, 10))

plt.title('Order\_Demand Trend')

plt.ylabel('Number of Order\_Demand')

plt.xlabel('Years')

plt.show()

colors\_list = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink', 'red']

explode\_list = [0.2, 0, 0, 0, 0, 0, 0.2] # ratio for each year with which to offset each wedge.

df2['Total'].plot(kind='pie',

figsize=(15, 6),

autopct='%1.1f%%',

startangle=90,

shadow=True,

labels=None, # turn off labels on pie chart

pctdistance=1.12, # the ratio between the center of each pie slice and the start of the text generated by autopct

colors=colors\_list, # add custom colors

explode=explode\_list

)

# scale the title up by 12% to match pctdistance

plt.title('Order\_Demand Trend [2011 - 2017]', y=1.12)

plt.axis('equal')

# add legend

plt.legend(labels=df2.index, loc='upper left')

plt.show()

---------------------------------------------------------------------------

KeyError Traceback (most recent call last)

/opt/conda/lib/python3.7/site-packages/pandas/core/indexes/base.py in get\_loc(self, key, method, tolerance)

3079 try:

-> 3080 return self.\_engine.get\_loc(casted\_key)

3081 except KeyError as err:

pandas/\_libs/index.pyx in pandas.\_libs.index.IndexEngine.get\_loc()

pandas/\_libs/index.pyx in pandas.\_libs.index.IndexEngine.get\_loc()

pandas/\_libs/hashtable\_class\_helper.pxi in pandas.\_libs.hashtable.PyObjectHashTable.get\_item()

pandas/\_libs/hashtable\_class\_helper.pxi in pandas.\_libs.hashtable.PyObjectHashTable.get\_item()

KeyError: 'Total'

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)

<ipython-input-17-c0a29ccb4c50> in <module>

2 explode\_list = [0.2, 0, 0, 0, 0, 0, 0.2] # ratio for each year with which to offset each wedge.

3

----> 4 df2['Total'].plot(kind='pie',

5 figsize=(15, 6),

6 autopct='%1.1f%%',

/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py in \_\_getitem\_\_(self, key)

3022 if self.columns.nlevels > 1:

3023 return self.\_getitem\_multilevel(key)

-> 3024 indexer = self.columns.get\_loc(key)

3025 if is\_integer(indexer):

3026 indexer = [indexer]

/opt/conda/lib/python3.7/site-packages/pandas/core/indexes/base.py in get\_loc(self, key, method, tolerance)

3080 return self.\_engine.get\_loc(casted\_key)

3081 except KeyError as err:

-> 3082 raise KeyError(key) from err

3083

3084 if tolerance is not None:

KeyError: 'Total'

rcParams['figure.figsize']=20,5 #Figure Size in Inches for Plotting

f, axes = plt.subplots(1,2)

normalDW=sn.boxplot(df['Warehouse'],df['Order\_Demand'],ax=axes[0]) #Create a variable for Regular Data for WH and OD

logWH=sn.boxplot(df['Warehouse'],np.log1p(df['Order\_Demand']),ax=axes[1]) #Craete a Variable with Log Transformation

/opt/conda/lib/python3.7/site-packages/seaborn/\_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

FutureWarning

/opt/conda/lib/python3.7/site-packages/seaborn/\_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

FutureWarning

df=df.groupby('Date')['Order\_Demand'].sum().reset\_index()

#Step-02: Indexing the Date Column as for further procssing.

df = df.set\_index('Date')

df.index #Lets check the index

#Step-03:#Averages daily sales value for the month, and we are using the start of each month as the timestamp.

monthly\_avg\_sales = df['Order\_Demand'].resample('MS').mean()

#In case there are Null values, they can be imputed using bfill.

monthly\_avg\_sales = monthly\_avg\_sales.fillna(monthly\_avg\_sales.bfill())

#Visualizing time series.

monthly\_avg\_sales.plot(figsize=(20,10))

plt.show()

Displaying the trends with their seasons

#Using Time Series for Decomposition.

from pylab import rcParams

import statsmodels.api as sm

rcParams['figure.figsize'] = 20, 10

decomposition = sm.tsa.seasonal\_decompose(monthly\_avg\_sales, model='additive')

fig = decomposition.plot()

plt.show()

df.head()

Order\_Demand

Date

2011-01-08 2

2011-05-31 108

2011-06-24 92000

2011-09-02 1250

2011-09-27 28

df2.head()

Warehouse Whse\_A Whse\_C Whse\_J Whse\_S

Year

2011 138.0 135.0 193.0 174.0

2012 28218.0 6889.0 150013.0 18515.0

2013 33607.0 7934.0 156251.0 20506.0

2014 28124.0 8187.0 157786.0 22307.0

2015 26686.0 8963.0 153937.0 20075.0

Creating the ARIMA Model

import itertools

p = d = q = range(0, 2)

pdq = list(itertools.product(p, d, q))

seasonal\_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]

#print('Examples of parameter combinations for Seasonal ARIMA...')

print('SARIMAX1: {} x {}'.format(pdq[1], seasonal\_pdq[1]))

print('SARIMAX2: {} x {}'.format(pdq[1], seasonal\_pdq[2]))

print('SARIMAX3: {} x {}'.format(pdq[2], seasonal\_pdq[3]))

print('SARIMAX4: {} x {}'.format(pdq[2], seasonal\_pdq[4]))

#STEP-02:

#Get the best params for the data. Choose the lowest AIC.

# The Akaike information criterion (AIC) is an estimator of the relative quality of statistical models for a

# given set of data.

# AIC measures how well a model fits the data while taking into account the overall complexity of the model.

# Large AIC: Model fits very well using a lot of features.

# Small AIC: Model fits similar fit but using lesser features.

# Hence LOWER THE AIC, the better it is.

#The code tests the given params using sarimax and outputs the AIC scores.

for param in pdq:

for param\_seasonal in seasonal\_pdq:

try:

mod = sm.tsa.statespace.SARIMAX(monthly\_avg\_sales,

order=param,

seasonal\_order=param\_seasonal,enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

print('SARIMA{}x{}12 - AIC:{}'.format(param, param\_seasonal, results.aic))

except:

continue

SARIMAX1: (0, 0, 1) x (0, 0, 1, 12)

SARIMAX2: (0, 0, 1) x (0, 1, 0, 12)

SARIMAX3: (0, 1, 0) x (0, 1, 1, 12)

SARIMAX4: (0, 1, 0) x (1, 0, 0, 12)

SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:2344.0704307124033

SARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1934.1928835402202

SARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1862.6079313938828

SARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1413.1955930564604

SARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1897.8374721163148

/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:568: ConvergenceWarning: Maximum Likelihood optimization failed to converge. Check mle\_retvals

ConvergenceWarning)

SARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1997.2321572752192

SARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:1440.6014550343189

SARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:1414.0578928049881

SARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:2265.276799776314

SARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1876.5613755956679

SARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:1790.8025464719374

SARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:1412.3527721130954

SARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:1934.8291713242158

SARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:1872.868645832047

SARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:1471.9391345294637

SARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:1414.2544026189005

SARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:2081.849011696963

SARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:1728.3815053402643

SARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:1741.6306755793414

SARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:1386.126892704832

SARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:1752.2949192972621

SARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:1727.5980407023796

SARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:1415.401329526097

SARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:1388.2877892398512

SARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:2055.0737963842494

SARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:1691.1145705442393

SARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:1713.7486773047408

SARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:1353.2955144788955

SARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:1754.9529578040397

SARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:1692.840451952837

SARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:1417.008047394343

SARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:1354.8462429513074

SARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:2111.4059905905533

SARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:1753.1287303174188

SARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:1769.1268218846342

SARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:1411.6074368683135

SARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:1753.617553918824

SARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:1755.128575013085

SARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:1411.6152646203473

SARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:1413.3192586311693

SARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:2084.8428767036157

SARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:1722.6088135545822

SARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:1740.7955949525992

SARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:1383.9622553576116

SARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:1755.828167065376

SARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:1724.4668991284752

SARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:1413.3086602327364

SARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:1385.5629811898293

SARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:2083.4881876767636

SARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:1722.6975835110582

SARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:1742.2670256724407

SARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:1386.1654409359567

SARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:1723.1897536728993

SARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:1724.4906930779784

SARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:1386.2231232636668

SARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:1387.9005061863727

SARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:2056.7044648338665

SARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:1692.7768596742885

SARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:1715.7178210620725

SARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:1351.1631068724378

SARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:1723.8391282464252

SARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:1694.437228788385

SARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:1385.749262474629

SARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:1336.8596923594766

from statsmodels.tsa.statespace.sarimax import SARIMAX

mod = sm.tsa.statespace.SARIMAX(monthly\_avg\_sales,

order=(1, 1, 1),

seasonal\_order=(0, 1, 1, 12),

enforce\_stationarity=False,

enforce\_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])

==============================================================================

coef std err z P>|z| [0.025 0.975]

------------------------------------------------------------------------------

ar.L1 -0.1130 0.828 -0.137 0.891 -1.736 1.510

ma.L1 -0.6431 0.621 -1.035 0.301 -1.861 0.575

ma.S.L12 -1.0802 0.028 -38.878 0.000 -1.135 -1.026

sigma2 3.708e+11 1.89e-12 1.96e+23 0.000 3.71e+11 3.71e+11

==============================================================================