In [246…

```
import pandas as pd
import numpy as np

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

from sklearn.impute import SimpleImputer

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn


from sklearn.linear_model import LinearRegression, Ridge, Lasso

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from scipy import stats

import math
```

In [247…

```
train = pd.read_csv('train_data.csv')
train.head(5)
print(train.shape)
```

```
(982644, 9)
C:\Users\2167419\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3444: DtypeW
arning: Columns (7) have mixed types.Specify dtype option on import or set low_memory=Fals
e.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [248…

```
train.head(5)
```

Out[248…

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	2	2015-06-30	5735	568	1	1	0	0
1	2	2	2015-06-30	9863	877	1	1	0	0
2	3	2	2015-06-30	13261	1072	1	1	0	1
3	4	2	2015-06-30	13106	1488	1	1	0	0
4	5	2	2015-06-30	6635	645	1	1	0	0

In [4]:

```
train['Open'].value_counts()
```

Out[4]:

```
1    814204
0    168440
Name: Open, dtype: int64
```

EDA

In [5]:

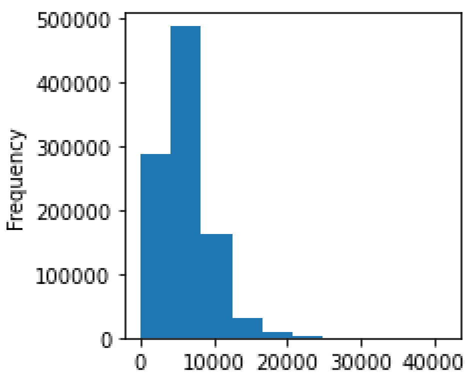
```
print(train['DayOfWeek'].unique())
print(train['Store'].nunique())
```

```
[2 1 7 6 5 4 3]
1115
```

In [6]:

```
#Perform an EDA (Exploratory Data Analysis) to see the impact of variables over Sales.


## distribution of sales price
c = lambda x: np.log2(x)

#train['log_sale'] = train['Sales'].apply(c)

fig = plt.figure(figsize =(3,3))
train['Sales'].plot(kind = 'hist')

## from the above plot it is being shown that data is positively skewed.We might take log

train2 = train[train['Sales']>0]
train2['log_sale'] = train2['Sales'].apply(c)
#train2['log_sale'].plot(kind = 'hist')
```

```
C:\Users\2167419\AppData\Local\Temp\1/ipykernel_18668/808680215.py:15: SettingWithCopyWarn
ing:
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_gu
ide/indexing.html#returning-a-view-versus-a-copy
  train2['log_sale'] = train2['Sales'].apply(c)
```



In [ ]:

```
## to see the impact of promo on sales:
plt.figure(figsize=(2,2))
print(train.columns)
import seaborn as sns
sns.boxplot(data = train, y = "Sales", x="Promo")
## from the box plot we can see promo impacts sales
```

In [ ]:

```
## to see the impact of SchoolHoliday on sales:
plt.figure(figsize=(2,2))
print(train.columns)
import seaborn as sns
sns.boxplot(data = train, y = "Sales", x="SchoolHoliday")
## from the box plot we can see SchoolHoliday does not impact sales
```

In [10]:

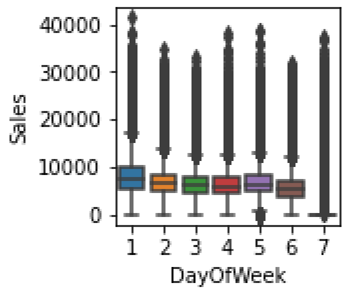
```
## to see the impact of DayOfWeek on sales:
plt.figure(figsize=(2,2))
print(train.columns)
import seaborn as sns
sns.boxplot(data = train, y = "Sales", x="DayOfWeek")
## from the box plot we can see DayOfWeek impacts sales value:
```

```
Index(['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
```

```
       'StateHoliday', 'SchoolHoliday'],
      dtype='object')
```

Out[10]:

```
<AxesSubplot:xlabel='DayOfWeek', ylabel='Sales'>
```



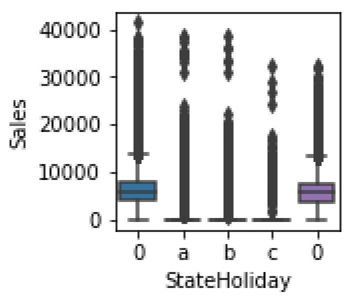
In [11]:

```
## to see the impact of StateHoliday on sales:
plt.figure(figsize=(2,2))
print(train.columns)
import seaborn as sns
sns.boxplot(data = train, y = "Sales", x="StateHoliday")
## from the box plot we can see StateHoliday somehow impacts sales value:
```

```
Index(['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
       'StateHoliday', 'SchoolHoliday'],
      dtype='object')
```

Out[11]:

```
<AxesSubplot:xlabel='StateHoliday', ylabel='Sales'>
```



Apply Linear Regression to predict the forecast :

In [ ]:

```
###Transform the variables by using data manipulation techniques like, One-Hot Encoding
print(train.columns)
```

In [ ]:

```
train['Store'].nunique()
```

In [19]:

```
## run the regression model for particular store as 1:
## in the same below way remaining store sales can be calculated.
## I did try having considered "store" as features but it was taking lot of time since the
## one ho encoding 1115 levels will be calculated...

print(train['Store'].value_counts().sort_values(ascending= False))

train1 = train[train['Store']==1]
train1.shape
```

```
1      911
261    911
248    911
249    911
250    911
```

```
        ...
712    727
711    727
710    727
348    727
155    727
Name: Store, Length: 1115, dtype: int64
```

Out[19]:

```
(911, 9)
```

In [20]:

```python
## feature selection:
train2= train1.drop(['Sales','Date','Open','Store'],axis =1)
print(train1.columns)
```

```
Index(['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
       'StateHoliday', 'SchoolHoliday'],
      dtype='object')
```

In [21]:

```python
## null values check and it shows no missing value in the train data :
train2.isnull().sum().sort_values(ascending = False)
```

Out[21]:

```
DayOfWeek        0
Customers        0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64
```

In [22]:

```python
train2.dtypes
```

Out[22]:

```
DayOfWeek         int64
Customers         int64
Promo             int64
StateHoliday     object
SchoolHoliday     int64
dtype: object
```

In [23]:

```python
#train1['Store'] = train1['Store'].astype(str)
train2['DayOfWeek'] = train2['DayOfWeek'].astype(str)
train2['Promo'] = train2['Promo'].astype(str)
train2['StateHoliday'] = train2['StateHoliday'].astype(str)
train2['SchoolHoliday'] = train2['SchoolHoliday'].astype(str)

#train1['DayOfWeek'] = str(train1['DayOfWeek'])
#train1['Promo'] = str(train1['Promo'])
#train1['StateHoliday'] = str(train1['StateHoliday'])
#train1['SchoolHoliday'] = str(train1['SchoolHoliday'])

train1.dtypes
```

Out[23]:

```
Store             int64
DayOfWeek         int64
Date             object
Sales             int64
Customers         int64
Open              int64
Promo             int64
StateHoliday     object
SchoolHoliday     int64
dtype: object
```

In [24]:

```python
train2.head(5)
```

Out[24]:

	DayOfWeek	Customers	Promo	StateHoliday	SchoolHoliday
0	2	568	1	0	0
1115	1	541	1	0	0
2230	7	0	0	0	0
3345	6	463	0	0	0
4460	5	420	0	0	0

In [25]:

```
from sklearn.preprocessing import OneHotEncoder
one_hot_encoded_data = pd.get_dummies(train2, columns =['DayOfWeek','Promo','SchoolHoliday
```

In [26]:

```
print(one_hot_encoded_data.shape)
one_hot_encoded_data.head(5)
```

```
(911, 16)
```

Out[26]:

	Customers	DayOfWeek_1	DayOfWeek_2	DayOfWeek_3	DayOfWeek_4	DayOfWeek_5	DayOfWeek_6	DayOfW
0	568	0	1	0	0	0	0	
1115	541	1	0	0	0	0	0	
2230	0	0	0	0	0	0	0	
3345	463	0	0	0	0	0	1	
4460	420	0	0	0	0	1	0	

In [27]:

```
from sklearn.model_selection import train_test_split
x = one_hot_encoded_data
y = train1['Sales']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=42)
```

In [28]:

```
## Train a single model for all stores, using storeId as a feature.
from sklearn.linear_model import LinearRegression

lm= LinearRegression()
model = lm.fit(x_train,y_train)

print(model.score(x_train,y_train)) # R Square or Coefficient of Determination
#print(model.intercept_)
#print(model.coef_)

#importance = model.coef_
# summarize feature importance
#for i,v in enumerate(importance):
#       print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
#plt.bar([x for x in range(len(importance))], importance)
#plt.show()
```

```
0.9850075207919289
```

In [30]:

```
## run the regession model for partiuclar store :
from sklearn.metrics import mean_squared_error
```

```
pred = lm.predict(x_test)
print(mean_squared_error(y_test,pred))
```

71294.38729382704

In [31]:

```
## Using relulariation techniques:
from sklearn.linear_model import Lasso, Ridge
lm = Lasso()
model = lm.fit(x_train,y_train)
print(model.score(x_train,y_train))
pred = lm.predict(x_test)
print(mean_squared_error(y_test,pred))

## it seeels ordiany regression gives better result:
```

0.9848637526262598
71940.55443439302

In [249…

```
## other regression techniques:
# When store is closed, sales = 0. Can this insight be used for Data Cleaning? Perform thi

# To answer this when store is closed we can impute by "simple imputer" method and run the

train_impute = pd.DataFrame(train['Sales'])

from sklearn.impute import SimpleImputer

numericimputer = SimpleImputer(missing_values= 0,strategy='mean')

numericdata= numericimputer.fit_transform(train_impute)

numericdata2 = pd.DataFrame(numericdata)

#numericdata2.rename(columns ={'0':'Sales'},inplace = True)

train_impute1 = pd.concat([train,numericdata2],axis =1)
train_impute1.rename(columns ={0:'Sales2'},inplace = True)
```

In [250…

```
train_impute1.head()
train_impute1 = train_impute1.sort_values(['Store','Date'])
train_impute1.head(5)
```

Out[250…

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	Sales2
981530	1	2	2013-01-01	0	0	0	0	a	1	6953.089734
980415	1	3	2013-01-02	5530	668	1	0	0	1	5530.000000
979300	1	4	2013-01-03	4327	578	1	0	0	1	4327.000000
978185	1	5	2013-01-04	4486	619	1	0	0	1	4486.000000
977070	1	6	2013-01-05	4997	635	1	0	0	1	4997.000000

In [10]:

```
## run the regression with the new data :

train1 = train_impute1[train_impute1['Store']==1]
```

```
train2= train1.drop(['Sales','Date','Customers','Open','Store','Sales2'],axis =1)

train2['DayOfWeek'] = train2['DayOfWeek'].astype(str)
train2['Promo'] =   train2['Promo'].astype(str)
train2['StateHoliday'] = train2['StateHoliday'].astype(str)
train2['SchoolHoliday'] = train2['SchoolHoliday'].astype(str)

from sklearn.preprocessing import OneHotEncoder
one_hot_encoded_data = pd.get_dummies(train2, columns =['DayOfWeek','Promo','SchoolHoliday

from sklearn.model_selection import train_test_split

x = one_hot_encoded_data
y = train1['Sales2']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=42)

from sklearn.linear_model import LinearRegression

lm= LinearRegression()
model = lm.fit(x_train,y_train)

print(model.score(x_train,y_train))

from sklearn.metrics import mean_squared_error
pred = lm.predict(x_test)
print(mean_squared_error(y_test,pred))

## here we can see MSE has been reduced after doing missing value imputation...
```

```
0.6616303420240928
641118.9971048271
```

In [126…

```
# Use Non-Linear Regressors like Random Forest or other Tree-based Regressors.

from sklearn.tree import DecisionTreeRegressor

tree = DecisionTreeRegressor(max_depth= 8)
model = tree.fit(x_train,y_train)

print(model.score(x_train,y_train))

pred = model.predict(x_test)
print(mean_squared_error(y_test,pred))

## By using tree based method mse has been more reduced and might be better model with ma
## it needs to be remember that Linear regression gave the beter r squared value but mse 
## dataset:
```

```
0.690699064243673
573915.7022067631
```

Time series mode by ARIMA:

In [11]:

```
## Time series mode by ARIMA:

train1 =  train_impute1[train_impute1['Store']==1]
train1_time = pd.DataFrame(train1[['Date','Sales2']])
#train1_time
train1_time.Date=pd.to_datetime(train1_time.Date)
train1_time
```

Out[11]:

	Date	Sales2
0	2015-06-30	5735.000000
1115	2015-06-29	5197.000000
2230	2015-06-28	6953.089734
3345	2015-06-27	4019.000000
4460	2015-06-26	3317.000000
...	...	...
977070	2013-01-05	4997.000000
978185	2013-01-04	4486.000000
979300	2013-01-03	4327.000000
980415	2013-01-02	5530.000000
981530	2013-01-01	6953.089734

911 rows × 2 columns

In [12]:

```
print(train1_time['Date'].max())
print(train1_time['Date'].min())
```

```
2015-06-30 00:00:00
2013-01-01 00:00:00
```

In [13]:

```
train1_time.sort_values(by = 'Date', ascending=True)
```

Out[13]:

	Date	Sales2
981530	2013-01-01	6953.089734
980415	2013-01-02	5530.000000
979300	2013-01-03	4327.000000
978185	2013-01-04	4486.000000
977070	2013-01-05	4997.000000
...	...	...
4460	2015-06-26	3317.000000
3345	2015-06-27	4019.000000
2230	2015-06-28	6953.089734
1115	2015-06-29	5197.000000
0	2015-06-30	5735.000000

911 rows × 2 columns

In [14]:

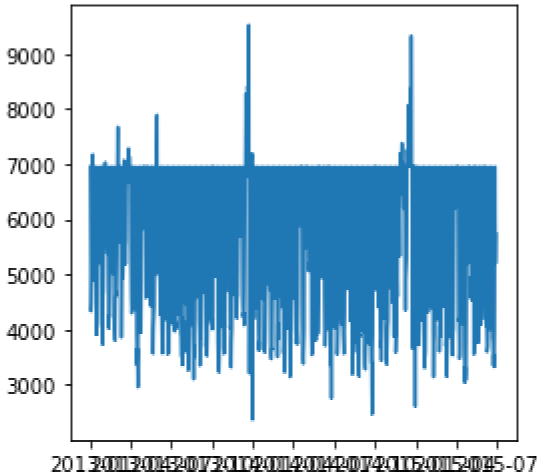
```
#!pip install pmdarima
#!pip3 install statsmodels
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
```

In [42]:

```
plt.figure(figsize=(4,4))
plt.plot(train1_time['Date'],train1_time['Sales2'])
```

Out[42]:

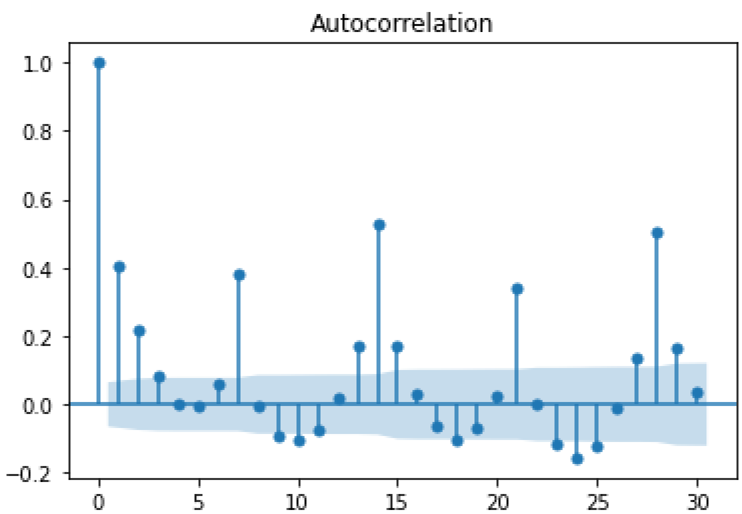
```
[<matplotlib.lines.Line2D at 0x27b6768c9d0>]
```



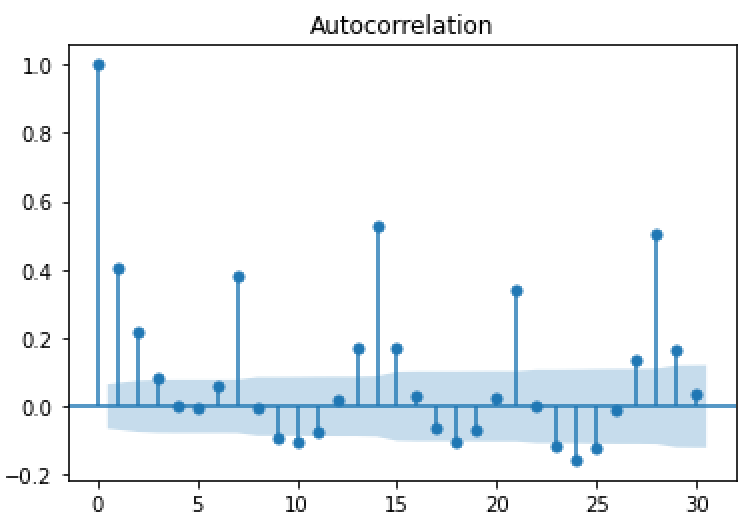
In [43]:

```
plt.figure(figsize=(3,3))
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
plot_acf(train1_time['Sales2'].dropna(),lags=30) # q=0
```

Out[43]:



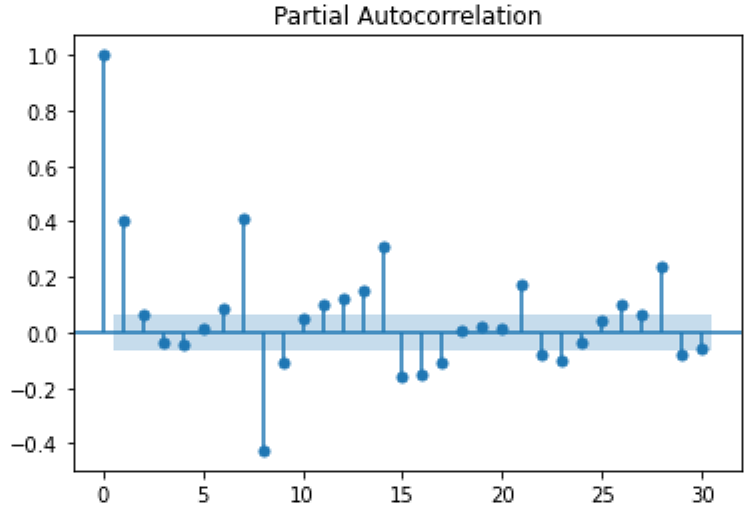

```
<Figure size 216x216 with 0 Axes>
```



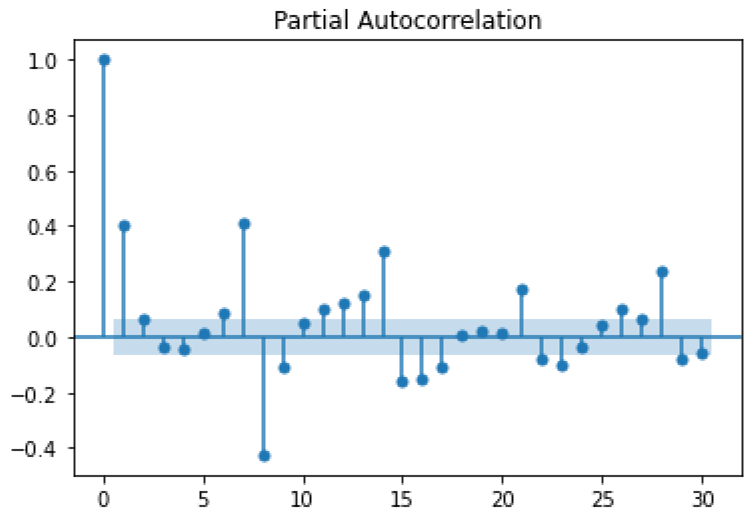
In [44]:

```
plt.figure(figsize=(3,3))
plot_pacf(train1_time['Sales2'].dropna(),lags=30) # p=0
```

Out[44]:




```
<Figure size 216x216 with 0 Axes>
```



In [45]:

```
#from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.arima_model import ARIMA
#!conda install pmdarima
```

In [18]:

```
import pmdarima as pm

train1_time_1 = pd.DataFrame(train1_time['Sales2'], columns =['Sales2'])
train1_time_1
```

Out[18]:

	Sales2
0	5735.000000
1115	5197.000000
2230	6953.089734
3345	4019.000000
4460	3317.000000
...	...
977070	4997.000000
978185	4486.000000
979300	4327.000000
980415	5530.000000

	Sales2
981530	6953.089734

911 rows × 1 columns

In [37]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train1_time_1 = pd.DataFrame(scaler.fit_transform(train1_time_1),columns=['Sales2'])
```

In [38]:
```
train1_time_1
```

Out[38]:

	Sales2
0	0.475052
1	0.041766
2	1.456060
3	-0.906955
4	-1.472321
...	...
906	-0.119307
907	-0.530849
908	-0.658902
909	0.309952
910	1.456060

911 rows × 1 columns

In [42]:
```
## divide the data into train and test data:

training_size=int(len(train1_time_1)*0.65)
print(training_size)
test_size= len(train1_time_1)-training_size
print(test_size)

train_data= train1_time_1.iloc[0:training_size]
test_data = train1_time_1.iloc[training_size:len(train1_time_1)]

print(train_data.shape)
print(test_data.shape)
#test_data= train1_time[0:training_size,:], train1_time[training_size:len(df1),:1]
```

```
592
319
(592, 1)
(319, 1)
```

In [40]:
```
train_data = train_data.reset_index()
```

In [43]:

```
train_data
```

Out[43]:

	Sales2
0	0.475052
1	0.041766
2	1.456060
3	-0.906955
4	-1.472321
...	...
587	-0.168435
588	-0.001724
589	0.629683
590	1.456060
591	-0.059710

592 rows × 1 columns

In [35]:

```
train1_time.dtypes
```

Out[35]:

```
Date      datetime64[ns]
Sales2           float64
dtype: object
```

In [44]:

```
model= pm.auto_arima(train_data.Sales2,
                      start_p=0,
                      start_q=0,
                     test='adf',
                     max_p=5,
                     max_q=5,
                     m=1,
                     d=None,
                     start_P=0,
                     D=0,
                     trace=True,
                     error_action='ignore',
                     suppress_warnings=True,
                     stepwise=True)
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1728.218, Time=0.02 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=1624.293, Time=0.05 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=1649.312, Time=0.06 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=1726.271, Time=0.00 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=1621.209, Time=0.07 sec
 ARIMA(3,0,0)(0,0,0)[0] intercept   : AIC=1623.107, Time=0.08 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=1623.159, Time=0.16 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=1621.835, Time=0.12 sec
 ARIMA(3,0,1)(0,0,0)[0] intercept   : AIC=1624.973, Time=0.26 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=1619.224, Time=0.03 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=1622.313, Time=0.03 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=1621.123, Time=0.04 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=1621.174, Time=0.10 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=1619.849, Time=0.06 sec
```

```
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=1622.990, Time=0.16 sec

Best model:  ARIMA(2,0,0)(0,0,0)[0]
Total fit time: 1.228 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  592
Model:               SARIMAX(2, 0, 0)   Log Likelihood                -806.612
Date:                Wed, 11 Jan 2023   AIC                           1619.224
Time:                        17:14:30   BIC                           1632.375
Sample:                             0   HQIC                          1624.346
                                - 592
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.3669      0.040      9.176      0.000       0.289       0.445
ar.L2          0.0926      0.036      2.578      0.010       0.022       0.163
sigma2         0.8930      0.058     15.444      0.000       0.780       1.006
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                40.53
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               0.94   Skew:                             0.64
Prob(H) (two-sided):                  0.68   Kurtosis:                         3.11
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

In [47]:

```python
n_periods= 319
test_predict=  model.predict(n_periods=n_periods,return_conf_int=False)
test_predict = pd.DataFrame(test_predict)
test_predict
```

Out[47]:

	0
0	1.128663e-01
1	3.588900e-02
2	2.361657e-02
3	1.198801e-02
4	6.584976e-03
...	...
314	3.531158e-86
315	1.902430e-86
316	1.024944e-86
317	5.521941e-87
318	2.974975e-87

319 rows × 1 columns

In [48]:

```python
test_data
```

Out[48]:

	Sales2
592	-0.909371

	Sales2
593	-1.087357
594	-1.477959
595	-0.908566
596	-0.910982
...	...
906	-0.119307
907	-0.530849
908	-0.658902
909	0.309952
910	1.456060

319 rows × 1 columns

In [49]:

```
import math
from sklearn.metrics import mean_squared_error
#print(math.sqrt(mean_squared_error(y_train,train_predict)))
### Test Data RMSE
print(math.sqrt(mean_squared_error(test_data['Sales2'],test_predict)))
```

```
0.925855490422089
```

In [ ]:

LSTM implementation :

In [ ]:

In [9]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input ,LSTM, Dense, Flatten,Dropout
from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import SGD, Adam
from sklearn.preprocessing import StandardScaler
```

In [10]:

```
train_impute1.head(2)
```

Out[10]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	Sales2
0	1	2	2015-06-30	5735	568	1	1	0	0	5735.0
1	2	2	2015-06-30	9863	877	1	1	0	0	9863.0

In [36]:

```
df = train_impute1[train_impute1['Store']==1]
```

In [37]:

```
df1=df.reset_index()['Sales2']
```

In [38]:
```
df1
```

Out[38]:
```
0      5735.000000
1      5197.000000
2      6953.089734
3      4019.000000
4      3317.000000
          ...
906    4997.000000
907    4486.000000
908    4327.000000
909    5530.000000
910    6953.089734
Name: Sales2, Length: 911, dtype: float64
```

In [39]:
```
from sklearn.preprocessing import MinMaxScaler
scaler= MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [43]:
```
df1.shape
print(len(df1))
```

```
911
```

In [46]:
```
##splitting dataset into train and test split
training_size=int(len(df1)*0.65)
test_size= len(df1)-training_size
train_data,test_data= df1[0:training_size,:], df1[training_size:len(df1),:1]
```

In [47]:
```
len(train_data)
```

Out[47]:
```
592
```

In [51]:
```
T = 5
D =1
X_train= []
y_train  =[]

for t in range(len(train_data)-T):
    x = train_data[t:t+T]
    X_train.append(x)
    y = train_data[t+T]
    y_train.append(y)
X_train = np.array(X_train).reshape(-1,T,1)
y_train = np.array(y_train)
N = len(X_train)
print("X_train.shape",X_train.shape,"Y_train.shape",y_train.shape)
```

```
X_train.shape (587, 5, 1) Y_train.shape (587, 1)
```

In [53]:
```
T = 5
D =1
X_test= []
ytest   =[]

for t in range(len(test_data)-T):
    x = test_data[t:t+T]
    X_test.append(x)
```

```
    y = test_data[t+T]
    ytest.append(y)
X_test = np.array(X_test).reshape(-1,T,1)
ytest = np.array(ytest)
N = len(X_test)
print("X_test.shape",X_test.shape,"Y_test.shape",ytest.shape)
```

```
X_test.shape (314, 5, 1) Y_test.shape (314, 1)
```

In [55]:

```
model = Sequential()

#model.add(LSTM(64,actvation= 'relu',input_shape = (X.shape[1]),X.shape[2]),return_sequenc
model.add(LSTM(64, activation='relu', return_sequences=True, input_shape=(5,1)))
model.add(LSTM(32, activation='relu', return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(1))

model.compile(optimizer='adam',loss= 'mse')

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 5, 64)             16896

 lstm_3 (LSTM)               (None, 32)                12416

 dropout_1 (Dropout)         (None, 32)                0

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
_________________________________________________________________
```

In [56]:

```
#fit = model.fit(X,Y,epochs=20,batch_size = 16,validation_split= 0.1)

model.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=100,batch_size=64,verbose=
```

```
Epoch 1/100
10/10 [==============================] - 3s 55ms/step - loss: 0.1506 - val_loss: 0.1139
Epoch 2/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0933 - val_loss: 0.0538
Epoch 3/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0417 - val_loss: 0.0290
Epoch 4/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0412 - val_loss: 0.0269
Epoch 5/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0345 - val_loss: 0.0286
Epoch 6/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0354 - val_loss: 0.0270
Epoch 7/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0348 - val_loss: 0.0265
Epoch 8/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0354 - val_loss: 0.0266
Epoch 9/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0340 - val_loss: 0.0265
Epoch 10/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0351 - val_loss: 0.0264
```

```
Epoch 11/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0338 - val_loss: 0.0262
Epoch 12/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0355 - val_loss: 0.0265
Epoch 13/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0345 - val_loss: 0.0260
Epoch 14/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0347 - val_loss: 0.0260
Epoch 15/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0346 - val_loss: 0.0258
Epoch 16/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0320 - val_loss: 0.0259
Epoch 17/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0331 - val_loss: 0.0257
Epoch 18/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0329 - val_loss: 0.0259
Epoch 19/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0320 - val_loss: 0.0257
Epoch 20/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0337 - val_loss: 0.0259
Epoch 21/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0335 - val_loss: 0.0255
Epoch 22/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0328 - val_loss: 0.0257
Epoch 23/100
10/10 [==============================] - 0s 12ms/step - loss: 0.0325 - val_loss: 0.0251
Epoch 24/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0324 - val_loss: 0.0252
Epoch 25/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0327 - val_loss: 0.0265
Epoch 26/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0326 - val_loss: 0.0249
Epoch 27/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0323 - val_loss: 0.0249
Epoch 28/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0335 - val_loss: 0.0254
Epoch 29/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0324 - val_loss: 0.0248
Epoch 30/100
10/10 [==============================] - 0s 12ms/step - loss: 0.0326 - val_loss: 0.0251
Epoch 31/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0325 - val_loss: 0.0245
Epoch 32/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0305 - val_loss: 0.0247
Epoch 33/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0318 - val_loss: 0.0244
Epoch 34/100
10/10 [==============================] - 0s 12ms/step - loss: 0.0307 - val_loss: 0.0240
Epoch 35/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0311 - val_loss: 0.0240
Epoch 36/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0303 - val_loss: 0.0238
Epoch 37/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0314 - val_loss: 0.0233
Epoch 38/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0307 - val_loss: 0.0233
Epoch 39/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0303 - val_loss: 0.0232
Epoch 40/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0308 - val_loss: 0.0231
Epoch 41/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0287 - val_loss: 0.0233
Epoch 42/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0291 - val_loss: 0.0226
Epoch 43/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0300 - val_loss: 0.0227
```

```
Epoch 44/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0290 - val_loss: 0.0230
Epoch 45/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0299 - val_loss: 0.0226
Epoch 46/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0279 - val_loss: 0.0228
Epoch 47/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0278 - val_loss: 0.0219
Epoch 48/100
10/10 [==============================] - 0s 12ms/step - loss: 0.0277 - val_loss: 0.0216
Epoch 49/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0262 - val_loss: 0.0217
Epoch 50/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0270 - val_loss: 0.0220
Epoch 51/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0290 - val_loss: 0.0235
Epoch 52/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0285 - val_loss: 0.0216
Epoch 53/100
10/10 [==============================] - 0s 12ms/step - loss: 0.0267 - val_loss: 0.0214
Epoch 54/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0269 - val_loss: 0.0211
Epoch 55/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0267 - val_loss: 0.0216
Epoch 56/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0269 - val_loss: 0.0208
Epoch 57/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0265 - val_loss: 0.0204
Epoch 58/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0263 - val_loss: 0.0215
Epoch 59/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0261 - val_loss: 0.0203
Epoch 60/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0258 - val_loss: 0.0209
Epoch 61/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0250 - val_loss: 0.0198
Epoch 62/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0253 - val_loss: 0.0195
Epoch 63/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0249 - val_loss: 0.0195
Epoch 64/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0245 - val_loss: 0.0192
Epoch 65/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0251 - val_loss: 0.0193
Epoch 66/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0242 - val_loss: 0.0192
Epoch 67/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0240 - val_loss: 0.0193
Epoch 68/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0243 - val_loss: 0.0201
Epoch 69/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0245 - val_loss: 0.0196
Epoch 70/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0235 - val_loss: 0.0182
Epoch 71/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0240 - val_loss: 0.0183
Epoch 72/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0239 - val_loss: 0.0179
Epoch 73/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0230 - val_loss: 0.0183
Epoch 74/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0240 - val_loss: 0.0185
Epoch 75/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0235 - val_loss: 0.0176
Epoch 76/100
10/10 [==============================] - 0s 10ms/step - loss: 0.0240 - val_loss: 0.0182
```

```
Epoch 77/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0228 - val_loss: 0.0182
Epoch 78/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0230 - val_loss: 0.0179
Epoch 79/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0220 - val_loss: 0.0174
Epoch 80/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0232 - val_loss: 0.0180
Epoch 81/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0222 - val_loss: 0.0178
Epoch 82/100
10/10 [==============================] - 0s 11ms/step - loss: 0.0220 - val_loss: 0.0187
Epoch 83/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0229 - val_loss: 0.0175
Epoch 84/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0216 - val_loss: 0.0178
Epoch 85/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0210 - val_loss: 0.0172
Epoch 86/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0216 - val_loss: 0.0170
Epoch 87/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0203 - val_loss: 0.0176
Epoch 88/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0216 - val_loss: 0.0174
Epoch 89/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0203 - val_loss: 0.0183
Epoch 90/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0221 - val_loss: 0.0167
Epoch 91/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0208 - val_loss: 0.0167
Epoch 92/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0203 - val_loss: 0.0172
Epoch 93/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0211 - val_loss: 0.0169
Epoch 94/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0206 - val_loss: 0.0159
Epoch 95/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0195 - val_loss: 0.0163
Epoch 96/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0209 - val_loss: 0.0167
Epoch 97/100
10/10 [==============================] - 0s 9ms/step - loss: 0.0200 - val_loss: 0.0168
Epoch 98/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0207 - val_loss: 0.0173
Epoch 99/100
10/10 [==============================] - 0s 7ms/step - loss: 0.0213 - val_loss: 0.0164
Epoch 100/100
10/10 [==============================] - 0s 8ms/step - loss: 0.0201 - val_loss: 0.0165
```

Out[56]:

```
<keras.callbacks.History at 0x239c07affd0>
```

In [62]:

```python
### Lets Do the prediction and check performance metrics
train_predict= model.predict(X_train)
test_predict= model.predict(X_test)
```

```
19/19 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
```

In [64]:

```python
import math
from sklearn.metrics import mean_squared_error
print(math.sqrt(mean_squared_error(y_train,train_predict)))
### Test Data RMSE
print(math.sqrt(mean_squared_error(ytest,test_predict)))
```

```
0.13558654291202632
0.12851681147841199
```

In [65]:

```
##Transformback to original form
train_predict= scaler.inverse_transform(train_predict)
test_predict=  scaler.inverse_transform(test_predict)
```

In [67]:

```
#validation between arima and LSTM
## based on the validation between the two test dataset it has confirmed that LSTM gives u
## normal arima gives mse 0.028 where as LSTM gives MSE as 0.12 on the test dataset ,
```

Cluster stores using sales and customer visits as features.

In [11]:

```
train_impute1.head(5)
```

Out[11]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	Sales2
0	1	2	2015-06-30	5735	568	1	1	0	0	5735.0
1	2	2	2015-06-30	9863	877	1	1	0	0	9863.0
2	3	2	2015-06-30	13261	1072	1	1	0	1	13261.0
3	4	2	2015-06-30	13106	1488	1	1	0	0	13106.0
4	5	2	2015-06-30	6635	645	1	1	0	0	6635.0

In [21]:

```
#train['Store'].value_counts()
print(train_impute1['Store'].nunique)
```

```
<bound method IndexOpsMixin.nunique of 0             1
1           2
2           3
3           4
4           5
          ...
982639    1111
982640    1112
982641    1113
982642    1114
982643    1115
Name: Store, Length: 982644, dtype: int64>
```

In [43]:

```
train_impute2 = train_impute1[['Store','Customers']]

train_impute3 = pd.DataFrame(train_impute2.groupby('Store')['Customers'].mean()).reset_inc
print(train_impute3.shape)

train_impute3 = train_impute3.drop(['Store'],axis =1)

train_impute3
```

```
(1115, 2)
```

Out[43]:

	Customers
0	468.165752

	Customers
1	484.173436
2	621.154775
3	1099.131723
4	444.019759
...	...
1110	375.192097
1111	697.354555
1112	597.177827
1113	2652.572997
1114	357.110867

1115 rows × 1 columns

In [38]:

```
from sklearn.cluster import KMeans
```

In [50]:

```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
scaleddata= scaler.fit_transform(train_impute3)
scaleddata2 = pd.DataFrame(scaleddata,columns = train_impute3.columns)
scaleddata2
```

Out[50]:

	Customers	cluster
0	-0.505375	-0.613069
1	-0.455066	-0.613069
2	-0.024560	-0.613069
3	1.477631	1.385679
4	-0.581261	-0.613069
...	...	...
1110	-0.797573	-0.613069
1111	0.214922	1.385679
1112	-0.099915	-0.613069
1113	6.359801	3.384426
1114	-0.854399	-0.613069

1115 rows × 2 columns

In [56]:

```
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_scor
kmeansfinal= KMeans(n_clusters= 3,init='k-means++',random_state=100)

kmeansfinal.fit(scaleddata2)
#c = kmeansfinal.labels_
#kmeanspredict=kmeansmodel.predict(scaleddata2)
#pcadf2['cluster'] = kmeansfinal.labels_
```

```python
scaleddata2['cluster'] = kmeansfinal.labels_
train_impute3['cluster'] = kmeansfinal.labels_
train_impute3
#print(silhouette_score(pcadf2,kmeansfinal.labels_))
#print(davies_bouldin_score(car2,kmeansfinal.labels_))
```

Out[56]:

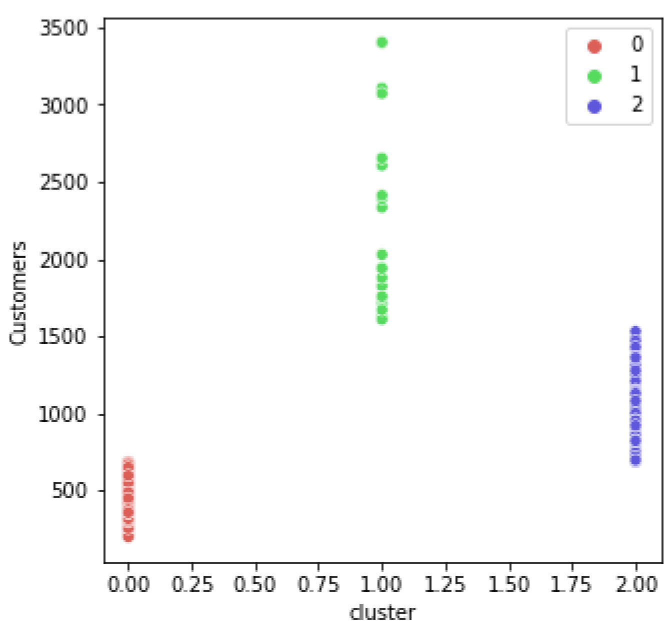
	Customers	cluster
0	468.165752	0
1	484.173436	0
2	621.154775	0
3	1099.131723	2
4	444.019759	0
...	...	...
1110	375.192097	0
1111	697.354555	2
1112	597.177827	0
1113	2652.572997	1
1114	357.110867	0

1115 rows × 2 columns

In [59]:

```python
import seaborn as sns
plt.figure(figsize=(5,5))
sns.scatterplot(x= train_impute3['cluster'], y=train_impute3['Customers'],
                hue= train_impute3.cluster.tolist(),
                palette=sns.color_palette("hls",3), legend='full', data= train_impute3)
#plt.title("Clusters for kmean clustering”)
```

Out[59]: <AxesSubplot:xlabel='cluster', ylabel='Customers'>



In [ ]:

```python
## here we can see 3 clusters have been crteated and we can have seperate prediction mode
```

ANN model with grid serach cv and k fold validation

In [251…

```
## applying ANN model for store no 1:


train1 = train_impute1[train_impute1['Store']==1]

train2= train1.drop(['Sales','Date','Open','Store','Sales2'],axis =1)

train2['DayOfWeek'] = train2['DayOfWeek'].astype(str)
train2['Promo'] =   train2['Promo'].astype(str)
train2['StateHoliday'] = train2['StateHoliday'].astype(str)
train2['SchoolHoliday'] = train2['SchoolHoliday'].astype(str)

from sklearn.preprocessing import OneHotEncoder
one_hot_encoded_data = pd.get_dummies(train2, columns =['DayOfWeek','Promo','SchoolHoliday

# from sklearn.model_selection import train_test_split

x = one_hot_encoded_data
y = train1['Sales2']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=42)

# from sklearn.linear_model import LinearRegression

# lm= LinearRegression()
# model = lm.fit(x_train,y_train)

# print(model.score(x_train,y_train))

# from sklearn.metrics import mean_squared_error
# pred = lm.predict(x_test)
# print(mean_squared_error(y_test,pred))
```

In [252…

```
one_hot_encoded_data
```

Out[252…

	Customers	DayOfWeek_1	DayOfWeek_2	DayOfWeek_3	DayOfWeek_4	DayOfWeek_5	DayOfWeek_6	DayO
981530	0	0	1	0	0	0	0	
980415	668	0	0	1	0	0	0	
979300	578	0	0	0	1	0	0	
978185	619	0	0	0	0	1	0	
977070	635	0	0	0	0	0	1	
...	...	...	...	...	...	...	...	
4460	420	0	0	0	0	1	0	
3345	463	0	0	0	0	0	1	
2230	0	0	0	0	0	0	0	
1115	541	1	0	0	0	0	0	
0	568	0	1	0	0	0	0	

911 rows × 16 columns

In [253…

```
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError
from keras.wrappers.scikit_learn import KerasRegressor
```

In [267…

```
model = Sequential()
model.add(Dense(32, kernel_initializer='normal', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64,  kernel_initializer='normal',activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64, kernel_initializer='normal', activation='relu'))
model.add(Dense(1,  kernel_initializer='normal', activation='linear'))
model.compile(loss= 'MeanSquaredLogarithmicError' ,
                  optimizer= 'rmsprop',

                  metrics= 'MeanSquaredLogarithmicError')
```

In [268…

```
history = model.fit(
    x_train,
    y_train,
    epochs=200,
    batch_size=64,
    validation_split=0)
```

```
Epoch 1/200
10/10 [==============================] - 0s 3ms/step - loss: 51.5231 - mean_squared_logari
thmic_error: 51.5231
Epoch 2/200
10/10 [==============================] - 0s 3ms/step - loss: 34.4315 - mean_squared_logari
thmic_error: 34.4315
Epoch 3/200
10/10 [==============================] - 0s 3ms/step - loss: 26.1298 - mean_squared_logari
thmic_error: 26.1298
Epoch 4/200
10/10 [==============================] - 0s 3ms/step - loss: 20.5983 - mean_squared_logari
thmic_error: 20.5983
Epoch 5/200
10/10 [==============================] - 0s 3ms/step - loss: 16.6172 - mean_squared_logari
thmic_error: 16.6172
Epoch 6/200
10/10 [==============================] - 0s 3ms/step - loss: 13.3514 - mean_squared_logari
thmic_error: 13.3514
Epoch 7/200
10/10 [==============================] - 0s 3ms/step - loss: 10.8371 - mean_squared_logari
thmic_error: 10.8371
Epoch 8/200
10/10 [==============================] - 0s 3ms/step - loss: 8.9055 - mean_squared_logarit
hmic_error: 8.9055
Epoch 9/200
10/10 [==============================] - 0s 2ms/step - loss: 7.3922 - mean_squared_logarit
hmic_error: 7.3922
Epoch 10/200
10/10 [==============================] - 0s 3ms/step - loss: 6.1077 - mean_squared_logarit
hmic_error: 6.1077
Epoch 11/200
10/10 [==============================] - 0s 4ms/step - loss: 5.1628 - mean_squared_logarit
hmic_error: 5.1628
Epoch 12/200
```

```
10/10 [==============================] - 0s 3ms/step - loss: 4.4035 - mean_squared_logarit
hmic_error: 4.4035
Epoch 13/200
10/10 [==============================] - 0s 2ms/step - loss: 3.8493 - mean_squared_logarit
hmic_error: 3.8493
Epoch 14/200
10/10 [==============================] - 0s 3ms/step - loss: 3.3773 - mean_squared_logarit
hmic_error: 3.3773
Epoch 15/200
10/10 [==============================] - 0s 3ms/step - loss: 3.0719 - mean_squared_logarit
hmic_error: 3.0719
Epoch 16/200
10/10 [==============================] - 0s 3ms/step - loss: 2.7910 - mean_squared_logarit
hmic_error: 2.7910
Epoch 17/200
10/10 [==============================] - 0s 2ms/step - loss: 2.5813 - mean_squared_logarit
hmic_error: 2.5813
Epoch 18/200
10/10 [==============================] - 0s 2ms/step - loss: 2.3158 - mean_squared_logarit
hmic_error: 2.3158
Epoch 19/200
10/10 [==============================] - 0s 2ms/step - loss: 2.0779 - mean_squared_logarit
hmic_error: 2.0779
Epoch 20/200
10/10 [==============================] - 0s 3ms/step - loss: 1.8759 - mean_squared_logarit
hmic_error: 1.8759
Epoch 21/200
10/10 [==============================] - 0s 2ms/step - loss: 1.6719 - mean_squared_logarit
hmic_error: 1.6719
Epoch 22/200
10/10 [==============================] - 0s 3ms/step - loss: 1.5089 - mean_squared_logarit
hmic_error: 1.5089
Epoch 23/200
10/10 [==============================] - 0s 3ms/step - loss: 1.3269 - mean_squared_logarit
hmic_error: 1.3269
Epoch 24/200
10/10 [==============================] - 0s 3ms/step - loss: 1.1712 - mean_squared_logarit
hmic_error: 1.1712
Epoch 25/200
10/10 [==============================] - 0s 3ms/step - loss: 1.0401 - mean_squared_logarit
hmic_error: 1.0401
Epoch 26/200
10/10 [==============================] - 0s 2ms/step - loss: 0.9227 - mean_squared_logarit
hmic_error: 0.9227
Epoch 27/200
10/10 [==============================] - 0s 3ms/step - loss: 0.7764 - mean_squared_logarit
hmic_error: 0.7764
Epoch 28/200
10/10 [==============================] - 0s 3ms/step - loss: 0.6787 - mean_squared_logarit
hmic_error: 0.6787
Epoch 29/200
10/10 [==============================] - 0s 3ms/step - loss: 0.6132 - mean_squared_logarit
hmic_error: 0.6132
Epoch 30/200
10/10 [==============================] - 0s 4ms/step - loss: 0.5261 - mean_squared_logarit
hmic_error: 0.5261
Epoch 31/200
10/10 [==============================] - 0s 3ms/step - loss: 0.4520 - mean_squared_logarit
hmic_error: 0.4520
Epoch 32/200
10/10 [==============================] - 0s 3ms/step - loss: 0.3850 - mean_squared_logarit
hmic_error: 0.3850
Epoch 33/200
10/10 [==============================] - 0s 2ms/step - loss: 0.3350 - mean_squared_logarit
hmic_error: 0.3350
Epoch 34/200
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.2893 - mean_squared_logarit
hmic_error: 0.2893
Epoch 35/200
10/10 [==============================] - 0s 3ms/step - loss: 0.2443 - mean_squared_logarit
hmic_error: 0.2443
Epoch 36/200
10/10 [==============================] - 0s 3ms/step - loss: 0.2104 - mean_squared_logarit
hmic_error: 0.2104
Epoch 37/200
10/10 [==============================] - 0s 3ms/step - loss: 0.1724 - mean_squared_logarit
hmic_error: 0.1724
Epoch 38/200
10/10 [==============================] - 0s 3ms/step - loss: 0.1388 - mean_squared_logarit
hmic_error: 0.1388
Epoch 39/200
10/10 [==============================] - 0s 3ms/step - loss: 0.1328 - mean_squared_logarit
hmic_error: 0.1328
Epoch 40/200
10/10 [==============================] - 0s 2ms/step - loss: 0.1076 - mean_squared_logarit
hmic_error: 0.1076
Epoch 41/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0827 - mean_squared_logarit
hmic_error: 0.0827
Epoch 42/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0809 - mean_squared_logarit
hmic_error: 0.0809
Epoch 43/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0653 - mean_squared_logarit
hmic_error: 0.0653
Epoch 44/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0596 - mean_squared_logarit
hmic_error: 0.0596
Epoch 45/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0511 - mean_squared_logarit
hmic_error: 0.0511
Epoch 46/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0506 - mean_squared_logarit
hmic_error: 0.0506
Epoch 47/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0444 - mean_squared_logarit
hmic_error: 0.0444
Epoch 48/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0462 - mean_squared_logarit
hmic_error: 0.0462
Epoch 49/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0465 - mean_squared_logarit
hmic_error: 0.0465
Epoch 50/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0376 - mean_squared_logarit
hmic_error: 0.0376
Epoch 51/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0435 - mean_squared_logarit
hmic_error: 0.0435
Epoch 52/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0436 - mean_squared_logarit
hmic_error: 0.0436
Epoch 53/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0359 - mean_squared_logarit
hmic_error: 0.0359
Epoch 54/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0405 - mean_squared_logarit
hmic_error: 0.0405
Epoch 55/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0341 - mean_squared_logarit
hmic_error: 0.0341
Epoch 56/200
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.0408 - mean_squared_logarit
hmic_error: 0.0408
Epoch 57/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0378 - mean_squared_logarit
hmic_error: 0.0378
Epoch 58/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0386 - mean_squared_logarit
hmic_error: 0.0386
Epoch 59/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0378 - mean_squared_logarit
hmic_error: 0.0378
Epoch 60/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0353 - mean_squared_logarit
hmic_error: 0.0353
Epoch 61/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0359 - mean_squared_logarit
hmic_error: 0.0359
Epoch 62/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0304 - mean_squared_logarit
hmic_error: 0.0304
Epoch 63/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0318 - mean_squared_logarit
hmic_error: 0.0318
Epoch 64/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0370 - mean_squared_logarit
hmic_error: 0.0370
Epoch 65/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0358 - mean_squared_logarit
hmic_error: 0.0358
Epoch 66/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0400 - mean_squared_logarit
hmic_error: 0.0400
Epoch 67/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0290 - mean_squared_logarit
hmic_error: 0.0290
Epoch 68/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0333 - mean_squared_logarit
hmic_error: 0.0333
Epoch 69/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0317 - mean_squared_logarit
hmic_error: 0.0317
Epoch 70/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0302 - mean_squared_logarit
hmic_error: 0.0302
Epoch 71/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0362 - mean_squared_logarit
hmic_error: 0.0362
Epoch 72/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0317 - mean_squared_logarit
hmic_error: 0.0317
Epoch 73/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0351 - mean_squared_logarit
hmic_error: 0.0351
Epoch 74/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0309 - mean_squared_logarit
hmic_error: 0.0309
Epoch 75/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0308 - mean_squared_logarit
hmic_error: 0.0308
Epoch 76/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0308 - mean_squared_logarit
hmic_error: 0.0308
Epoch 77/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0281 - mean_squared_logarit
hmic_error: 0.0281
Epoch 78/200
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.0313 - mean_squared_logarit
hmic_error: 0.0313
Epoch 79/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0281 - mean_squared_logarit
hmic_error: 0.0281
Epoch 80/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0337 - mean_squared_logarit
hmic_error: 0.0337
Epoch 81/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0294 - mean_squared_logarit
hmic_error: 0.0294
Epoch 82/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0308 - mean_squared_logarit
hmic_error: 0.0308
Epoch 83/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0334 - mean_squared_logarit
hmic_error: 0.0334
Epoch 84/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0334 - mean_squared_logarit
hmic_error: 0.0334
Epoch 85/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0338 - mean_squared_logarit
hmic_error: 0.0338
Epoch 86/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0308 - mean_squared_logarit
hmic_error: 0.0308
Epoch 87/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0310 - mean_squared_logarit
hmic_error: 0.0310
Epoch 88/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0339 - mean_squared_logarit
hmic_error: 0.0339
Epoch 89/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0266 - mean_squared_logarit
hmic_error: 0.0266
Epoch 90/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0284 - mean_squared_logarit
hmic_error: 0.0284
Epoch 91/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0319 - mean_squared_logarit
hmic_error: 0.0319
Epoch 92/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0276 - mean_squared_logarit
hmic_error: 0.0276
Epoch 93/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0287 - mean_squared_logarit
hmic_error: 0.0287
Epoch 94/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0332 - mean_squared_logarit
hmic_error: 0.0332
Epoch 95/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0285 - mean_squared_logarit
hmic_error: 0.0285
Epoch 96/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0283 - mean_squared_logarit
hmic_error: 0.0283
Epoch 97/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0252 - mean_squared_logarit
hmic_error: 0.0252
Epoch 98/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0318 - mean_squared_logarit
hmic_error: 0.0318
Epoch 99/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0296 - mean_squared_logarit
hmic_error: 0.0296
Epoch 100/200
```

```
10/10 [==============================] - 0s 3ms/step - loss: 0.0312 - mean_squared_logarit
hmic_error: 0.0312
Epoch 101/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0271 - mean_squared_logarit
hmic_error: 0.0271
Epoch 102/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0272 - mean_squared_logarit
hmic_error: 0.0272
Epoch 103/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0316 - mean_squared_logarit
hmic_error: 0.0316
Epoch 104/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0264 - mean_squared_logarit
hmic_error: 0.0264
Epoch 105/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0296 - mean_squared_logarit
hmic_error: 0.0296
Epoch 106/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0293 - mean_squared_logarit
hmic_error: 0.0293
Epoch 107/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0296 - mean_squared_logarit
hmic_error: 0.0296
Epoch 108/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0267 - mean_squared_logarit
hmic_error: 0.0267
Epoch 109/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0244 - mean_squared_logarit
hmic_error: 0.0244
Epoch 110/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0303 - mean_squared_logarit
hmic_error: 0.0303
Epoch 111/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0318 - mean_squared_logarit
hmic_error: 0.0318
Epoch 112/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0249 - mean_squared_logarit
hmic_error: 0.0249
Epoch 113/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0297 - mean_squared_logarit
hmic_error: 0.0297
Epoch 114/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0252 - mean_squared_logarit
hmic_error: 0.0252
Epoch 115/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0296 - mean_squared_logarit
hmic_error: 0.0296
Epoch 116/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0310 - mean_squared_logarit
hmic_error: 0.0310
Epoch 117/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0280 - mean_squared_logarit
hmic_error: 0.0280
Epoch 118/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0287 - mean_squared_logarit
hmic_error: 0.0287
Epoch 119/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0280 - mean_squared_logarit
hmic_error: 0.0280
Epoch 120/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0284 - mean_squared_logarit
hmic_error: 0.0284
Epoch 121/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0280 - mean_squared_logarit
hmic_error: 0.0280
Epoch 122/200
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.0261 - mean_squared_logarit
hmic_error: 0.0261
Epoch 123/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0271 - mean_squared_logarit
hmic_error: 0.0271
Epoch 124/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0264 - mean_squared_logarit
hmic_error: 0.0264
Epoch 125/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0286 - mean_squared_logarit
hmic_error: 0.0286
Epoch 126/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0279 - mean_squared_logarit
hmic_error: 0.0279
Epoch 127/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0287 - mean_squared_logarit
hmic_error: 0.0287
Epoch 128/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0237 - mean_squared_logarit
hmic_error: 0.0237
Epoch 129/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0255 - mean_squared_logarit
hmic_error: 0.0255
Epoch 130/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0258 - mean_squared_logarit
hmic_error: 0.0258
Epoch 131/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0297 - mean_squared_logarit
hmic_error: 0.0297
Epoch 132/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0261 - mean_squared_logarit
hmic_error: 0.0261
Epoch 133/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0240 - mean_squared_logarit
hmic_error: 0.0240
Epoch 134/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0292 - mean_squared_logarit
hmic_error: 0.0292
Epoch 135/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0258 - mean_squared_logarit
hmic_error: 0.0258
Epoch 136/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0269 - mean_squared_logarit
hmic_error: 0.0269
Epoch 137/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0261 - mean_squared_logarit
hmic_error: 0.0261
Epoch 138/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0300 - mean_squared_logarit
hmic_error: 0.0300
Epoch 139/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0274 - mean_squared_logarit
hmic_error: 0.0274
Epoch 140/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0236 - mean_squared_logarit
hmic_error: 0.0236
Epoch 141/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0289 - mean_squared_logarit
hmic_error: 0.0289
Epoch 142/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0294 - mean_squared_logarit
hmic_error: 0.0294
Epoch 143/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0257 - mean_squared_logarit
hmic_error: 0.0257
Epoch 144/200
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.0228 - mean_squared_logarit
hmic_error: 0.0228
Epoch 145/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0285 - mean_squared_logarit
hmic_error: 0.0285
Epoch 146/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0269 - mean_squared_logarit
hmic_error: 0.0269
Epoch 147/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0283 - mean_squared_logarit
hmic_error: 0.0283
Epoch 148/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0260 - mean_squared_logarit
hmic_error: 0.0260
Epoch 149/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0311 - mean_squared_logarit
hmic_error: 0.0311
Epoch 150/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0257 - mean_squared_logarit
hmic_error: 0.0257
Epoch 151/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0248 - mean_squared_logarit
hmic_error: 0.0248
Epoch 152/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0270 - mean_squared_logarit
hmic_error: 0.0270
Epoch 153/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0265 - mean_squared_logarit
hmic_error: 0.0265
Epoch 154/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0265 - mean_squared_logarit
hmic_error: 0.0265
Epoch 155/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0270 - mean_squared_logarit
hmic_error: 0.0270
Epoch 156/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0302 - mean_squared_logarit
hmic_error: 0.0302
Epoch 157/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0243 - mean_squared_logarit
hmic_error: 0.0243
Epoch 158/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0293 - mean_squared_logarit
hmic_error: 0.0293
Epoch 159/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0220 - mean_squared_logarit
hmic_error: 0.0220
Epoch 160/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0243 - mean_squared_logarit
hmic_error: 0.0243
Epoch 161/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0247 - mean_squared_logarit
hmic_error: 0.0247
Epoch 162/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0313 - mean_squared_logarit
hmic_error: 0.0313
Epoch 163/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0276 - mean_squared_logarit
hmic_error: 0.0276
Epoch 164/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0253 - mean_squared_logarit
hmic_error: 0.0253
Epoch 165/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0258 - mean_squared_logarit
hmic_error: 0.0258
Epoch 166/200
```

```
10/10 [==============================] - 0s 3ms/step - loss: 0.0268 - mean_squared_logarit
hmic_error: 0.0268
Epoch 167/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0276 - mean_squared_logarit
hmic_error: 0.0276
Epoch 168/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0276 - mean_squared_logarit
hmic_error: 0.0276
Epoch 169/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0263 - mean_squared_logarit
hmic_error: 0.0263
Epoch 170/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0239 - mean_squared_logarit
hmic_error: 0.0239
Epoch 171/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0253 - mean_squared_logarit
hmic_error: 0.0253
Epoch 172/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0247 - mean_squared_logarit
hmic_error: 0.0247
Epoch 173/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0268 - mean_squared_logarit
hmic_error: 0.0268
Epoch 174/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0240 - mean_squared_logarit
hmic_error: 0.0240
Epoch 175/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0253 - mean_squared_logarit
hmic_error: 0.0253
Epoch 176/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0223 - mean_squared_logarit
hmic_error: 0.0223
Epoch 177/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0254 - mean_squared_logarit
hmic_error: 0.0254
Epoch 178/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0296 - mean_squared_logarit
hmic_error: 0.0296
Epoch 179/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0240 - mean_squared_logarit
hmic_error: 0.0240
Epoch 180/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0275 - mean_squared_logarit
hmic_error: 0.0275
Epoch 181/200
10/10 [==============================] - 0s 2ms/step - loss: 0.0245 - mean_squared_logarit
hmic_error: 0.0245
Epoch 182/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0271 - mean_squared_logarit
hmic_error: 0.0271
Epoch 183/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0253 - mean_squared_logarit
hmic_error: 0.0253
Epoch 184/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0246 - mean_squared_logarit
hmic_error: 0.0246
Epoch 185/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0246 - mean_squared_logarit
hmic_error: 0.0246
Epoch 186/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0285 - mean_squared_logarit
hmic_error: 0.0285
Epoch 187/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0241 - mean_squared_logarit
hmic_error: 0.0241
Epoch 188/200
```

```
10/10 [==============================] - 0s 3ms/step - loss: 0.0283 - mean_squared_logarit
hmic_error: 0.0283
Epoch 189/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0256 - mean_squared_logarit
hmic_error: 0.0256
Epoch 190/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0265 - mean_squared_logarit
hmic_error: 0.0265
Epoch 191/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0242 - mean_squared_logarit
hmic_error: 0.0242
Epoch 192/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0257 - mean_squared_logarit
hmic_error: 0.0257
Epoch 193/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0247 - mean_squared_logarit
hmic_error: 0.0247
Epoch 194/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0240 - mean_squared_logarit
hmic_error: 0.0240
Epoch 195/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0242 - mean_squared_logarit
hmic_error: 0.0242
Epoch 196/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0254 - mean_squared_logarit
hmic_error: 0.0254
Epoch 197/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0249 - mean_squared_logarit
hmic_error: 0.0249
Epoch 198/200
10/10 [==============================] - 0s 4ms/step - loss: 0.0274 - mean_squared_logarit
hmic_error: 0.0274
Epoch 199/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0225 - mean_squared_logarit
hmic_error: 0.0225
Epoch 200/200
10/10 [==============================] - 0s 3ms/step - loss: 0.0283 - mean_squared_logarit
hmic_error: 0.0283
```

In [269…

```python
from sklearn.metrics import mean_squared_error ,mean_absolute_error
pred = model.predict(x_test)
print(mean_squared_error(y_test,pred))

## from the normal regression model we gor mse as "641118.9971048271" but by ANN we got MS
## we got the best result...
```

```
9/9 [==============================] - 0s 1ms/step
82534.58442302846
```

In [230…

```python
from sklearn.metrics import mean_squared_error ,mean_absolute_error
pred = grid_results.predict(x_test)
print(mean_squared_error(y_test,pred))
pred.shape
## from the normal regression model we gor mse as "641118.9971048271" but by ANN we got MS
## we got the best result...
```

```
8895786.312789395
```

Out[230…

```
(274,)
```

In [272…

```python
def create_model(optimizer='rmsprop'):
    model = Sequential()
    model.add(Dense(32, kernel_initializer='normal', activation='relu'))
    model.add(Dropout(0.2))
```

```python
    model.add(Dense(64,  kernel_initializer='normal',activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(64, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1,  kernel_initializer='normal', activation='linear'))
    model.compile(loss= 'MeanSquaredLogarithmicError' ,
                  optimizer= 'adam',
                  metrics= 'MeanSquaredLogarithmicError')
    return model

model = KerasRegressor( build_fn = create_model,
                        epochs=40,
                        batch_size=64,
                        verbose=0)
#results = cross_val_score(model, x_train, y_train,cv =5)
#results
param_grid = {'optimizer':  ['rmsprop'],
              'batch_size': [64],
              'epochs' : [200]
             #'learning_rate': [0.001, 0.01, 0.1, 0.2, 0.3]
             }

grid = GridSearchCV(model,
                    param_grid=param_grid,cv=3)

grid_results = grid.fit(x_train,y_train)
grid_results
print("Best: %f using %s" % (grid_results.cv_results_['mean_test_score'], grid_results.bes

#means = grid_results.cv_results_['MeanSquaredLogarithmicError']
#grid_results.cv_results
print(means)
# model.compile(loss= 'MeanSquaredLogarithmicError' ,
#                  optimizer= Adam(learning_rate=0.01),
#                  metrics= 'MeanSquaredLogarithmicError')
```

```
C:\Users\2167419\AppData\Local\Temp\1/ipykernel_27352/1938646446.py:14: DeprecationWarnin
g: KerasRegressor is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) inst
ead. See https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  model = KerasRegressor( build_fn = create_model,
Best: -0.079464 using {'batch_size': 64, 'epochs': 200, 'optimizer': 'rmsprop'}
[-1.99220061]
```

In [273…

```python
from sklearn.metrics import mean_squared_error ,mean_absolute_error
pred = grid_results.predict(x_test)
print(mean_squared_error(y_test,pred))
pred.shape
## from the normal regression model we gor mse as "641118.9971048271" but by ANN we got MS
## we got the best result...
```

```
300715.30087884166
```

Out[273…

```
(274,)
```

In [281…

```python
## run "jupyter nbconvert --to webpdf --allow-chromium-download Untitled.ipynb" where pyth
```

In [ ]: