Data

```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Importing Data
URL = '/content/ECP_DataSet.csv'
df = pd.read_csv(URL)
```

Data Exploration

```
print("Return first 5 rows.","\n")
df.head()
```

```
Return first 5 rows.
```

	Start time UTC	End time UTC	Start time UTC+03:00	End time UTC+03:00	Electricity consumption in Finland
0	2015-12-31 21:00:00	2015-12-31 22:00:00	2016-01-01 00:00:00	2016-01-01 01:00:00	10800.0
1	2015-12-31 22:00:00	2015-12-31 23:00:00	2016-01-01 01:00:00	2016-01-01 02:00:00	10431.0
2	2015-12-31 23:00:00	2016-01-01 00:00:00	2016-01-01 02:00:00	2016-01-01 03:00:00	10005.0

```
print("Return last 5 rows.","\n")
df.tail()
```

```
Return last 5 rows.
```

	Start time UTC	End time UTC	Start time UTC+03:00	End time UTC+03:00	Electricity consumption in Finland
52961	2021-12-31 16:00:00	2021-12-31 17:00:00	2021-12-31 19:00:00	2021-12-31 20:00:00	11447.0
52962	2021-12-31 17:00:00	2021-12-31 18:00:00	2021-12-31 20:00:00	2021-12-31 21:00:00	11237.0
52963	2021-12-31 18:00:00	2021-12-31 19:00:00	2021-12-31 21:00:00	2021-12-31 22:00:00	10914.0
52964	2021-12-31	2021-12-31	2021-12-31	2021-12-31	10599.0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 52966 entries, 0 to 52965
Data columns (total 5 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Start time UTC                     52966 non-null  object
 1   End time UTC                       52966 non-null  object
 2   Start time UTC+03:00               52966 non-null  object
 3   End time UTC+03:00                 52966 non-null  object
 4   Electricity consumption in Finland  52966 non-null  float64
```

```
dtypes: float64(1), object(4)
memory usage: 2.0+ MB
```

```
those that summarize the central tendency, dispersion and shape of a dataset’s distribution, excluding NaN values.", "\n")
```

```
       Electricity consumption in Finland
count                        52966.000000
mean                          9488.750519
std                           1576.241673
min                           5341.000000
25%                           8322.000000
50%                           9277.000000
75%                          10602.000000
max                          15105.000000
```

Feature Extraction

```
del df["Start time UTC"]
del df["End time UTC"]
del df["Start time UTC+03:00"]
df.rename(columns={"End time UTC+03:00":"DateTime","Electricity consumption in Finland":"Consumption"},inplace=True)
print(df.head(5))
```

```
              DateTime  Consumption
0  2016-01-01 01:00:00      10800.0
1  2016-01-01 02:00:00      10431.0
2  2016-01-01 03:00:00      10005.0
3  2016-01-01 04:00:00       9722.0
4  2016-01-01 05:00:00       9599.0
```

since we are dealing with time series data we should edite the index from 1 2 3 ... –> DateTime format.

```
dataset = df
dataset["Month"] = pd.to_datetime(df["DateTime"]).dt.month
dataset["Year"] = pd.to_datetime(df["DateTime"]).dt.year
dataset["Date"] = pd.to_datetime(df["DateTime"]).dt.date
dataset["Time"] = pd.to_datetime(df["DateTime"]).dt.time
dataset["Week"] = pd.to_datetime(df["DateTime"]).dt.week
dataset["Day"] = pd.to_datetime(df["DateTime"]).dt.day_name()
dataset = df.set_index("DateTime")
dataset.index = pd.to_datetime(dataset.index)
```

```
<ipython-input-9-9956197795b3>:6: FutureWarning: Series.dt.weekofyear and Series.dt.week have been deprecated. Please
  dataset["Week"] = pd.to_datetime(df["DateTime"]).dt.week
```

```
dataset.head()
```

DateTime	Consumption	Month	Year	Date	Time	Week	Day
2016-01-01 01:00:00	10800.0	1	2016	2016-01-01	01:00:00	53	Friday
2016-01-01 02:00:00	10431.0	1	2016	2016-01-01	02:00:00	53	Friday
2016-01-01 03:00:00	10005.0	1	2016	2016-01-01	03:00:00	53	Friday
2016-01-01 04:00:00	9722.0	1	2016	2016-01-01	04:00:00	53	Friday
2016-01-01 05:00:00	9599.0	1	2016	2016-01-01	05:00:00	53	Friday

```
print("")
print("Total Number of Years: ", dataset.Year.nunique() )
print(dataset.Year.unique())
```

```
Total Number of Years:  7
[2016 2017 2018 2019 2020 2021 2022]
```

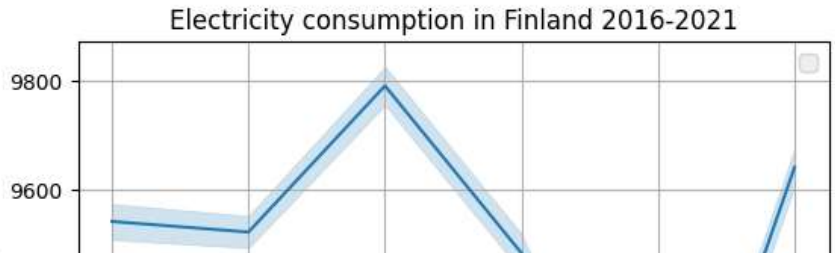
```
# By assuming week starts on Mondey and ends on Sunday.
# The closest start would be on Monday 4-1-2016
# The closest end would be on sunday 26-12-2021
# So omit first 71 rows and last 121 rows.
dataset = dataset[71:-121]
dataset.tail()
```

DateTime	Consumption	Month	Year	Date	Time	Week	Day
2021-12-26 19:00:00	12550.0	12	2021	2021-12-26	19:00:00	51	Sunday
2021-12-26 20:00:00	12622.0	12	2021	2021-12-26	20:00:00	51	Sunday
2021-12-26 21:00:00	12574.0	12	2021	2021-12-26	21:00:00	51	Sunday
2021-12-26 22:00:00	12384.0	12	2021	2021-12-26	22:00:00	51	Sunday
2021-12-26 23:00:00	12044.0	12	2021	2021-12-26	23:00:00	51	Sunday

Data Visualizations

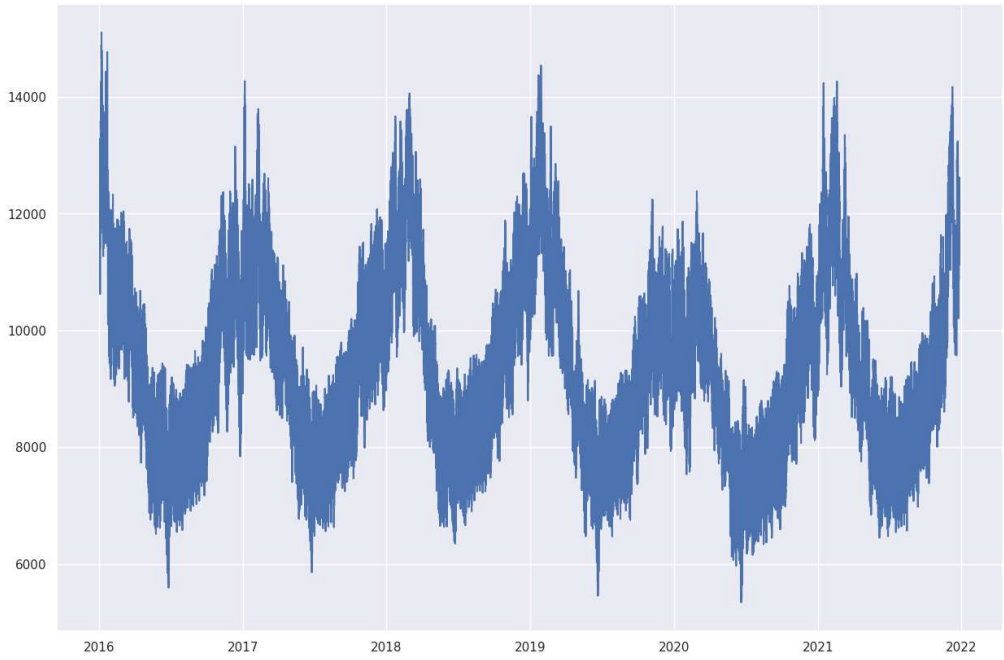
```
from matplotlib import style
fig = plt.figure()
axes1 = plt.subplot2grid((1,1), (0,0))

style.use("ggplot")
sns.lineplot(x= dataset["Year"], y= dataset["Consumption"], data = dataset)
sns.set(rc={'figure.figsize': (20,10)})

plt.title("Electricity consumption in Finland 2016-2021")
plt.xlabel("Date")
plt.ylabel("Energy in MW")
plt.grid(True)
plt.legend()

for label in axes1.xaxis.get_ticklabels():
    label.set_rotation(90)
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that art
```



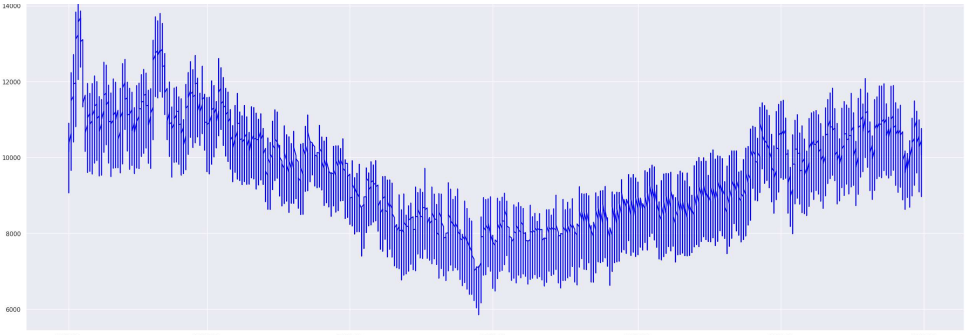

```
plt.figure(figsize=(15,10))
plt.plot(dataset["Consumption"])
```

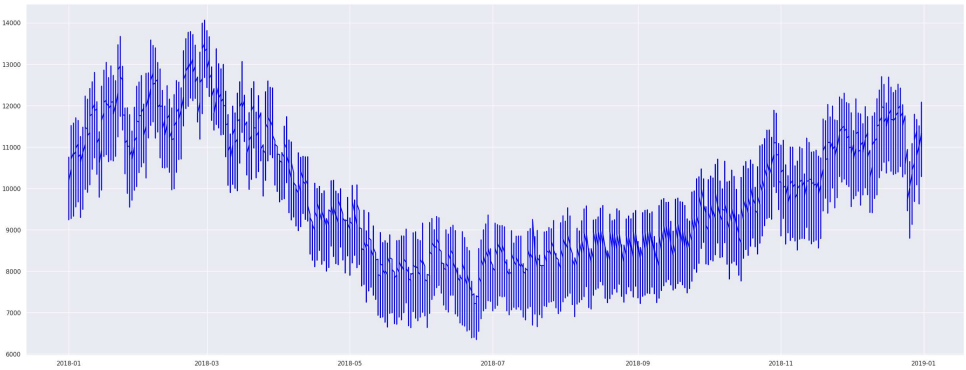
```
[<matplotlib.lines.Line2D at 0x7f7827c67a90>]
```

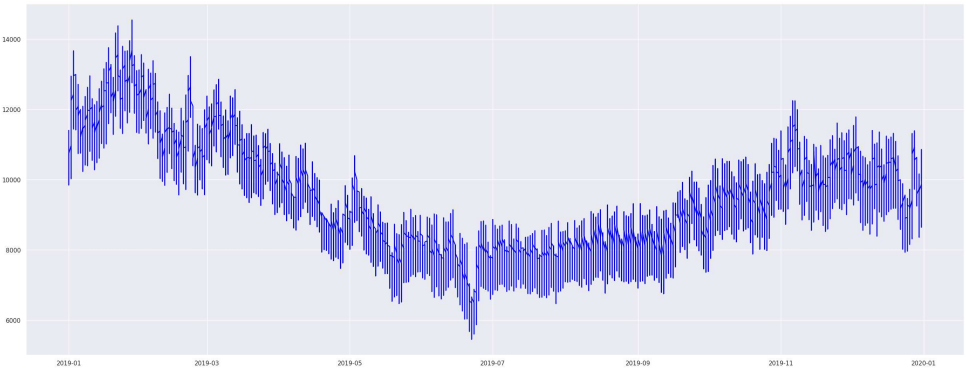


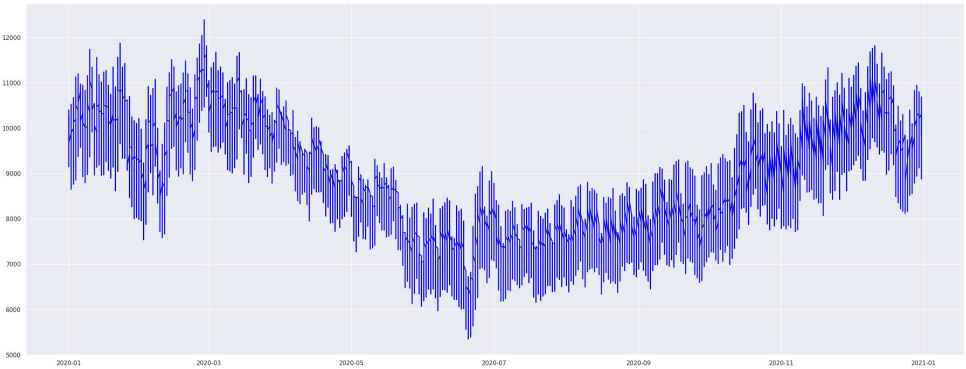

```
# Energy Consumption Each Year
from matplotlib import style

fig = plt.figure(figsize = (30,30))

ax1 = fig.add_subplot(611)
ax2 = fig.add_subplot(612)
ax3 = fig.add_subplot(613)
ax4 = fig.add_subplot(614)
ax5 = fig.add_subplot(615)
ax6 = fig.add_subplot(616)

style.use("ggplot")

y_2016 = dataset.loc["2016"]["Consumption"].to_list()
x_2016 = dataset.loc["2016"]["Date"].to_list()
ax1.plot(x_2016, y_2016, color= "blue", linewidth= 1.7)
```

```
y_2017 = dataset.loc["2017"]["Consumption"].to_list()
x_2017 = dataset.loc["2017"]["Date"].to_list()
ax2.plot(x_2017, y_2017, color= "blue", linewidth= 1.7)

y_2018 = dataset.loc["2018"]["Consumption"].to_list()
x_2018 = dataset.loc["2018"]["Date"].to_list()
ax3.plot(x_2018, y_2018, color= "blue", linewidth= 1.7)

y_2019 = dataset.loc["2019"]["Consumption"].to_list()
x_2019 = dataset.loc["2019"]["Date"].to_list()
ax4.plot(x_2019, y_2019, color= "blue", linewidth= 1.7)

y_2020 = dataset.loc["2020"]["Consumption"].to_list()
x_2020 = dataset.loc["2020"]["Date"].to_list()
ax5.plot(x_2020, y_2020, color= "blue", linewidth= 1.7)

y_2021 = dataset.loc["2021"]["Consumption"].to_list()
x_2021 = dataset.loc["2021"]["Date"].to_list()
ax6.plot(x_2021, y_2021, color= "blue", linewidth= 1.7)

plt.rcParams["figure.figsize"] = (30, 15)
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=2.5, wspace=0.4, hspace=0.4)
plt.title("Energy Consumption")
plt.xlabel("Date")
plt.ylabel("Energy in MW")
plt.grid(True, alpha=1)
plt.legend()
```

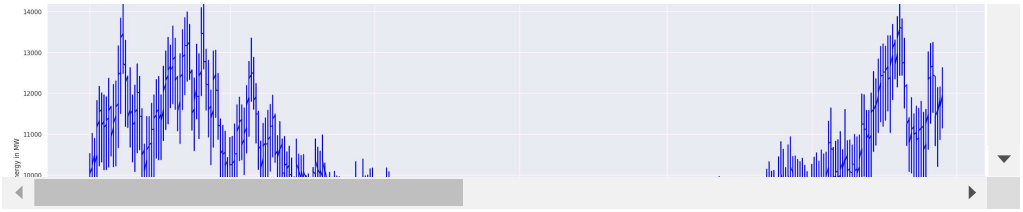






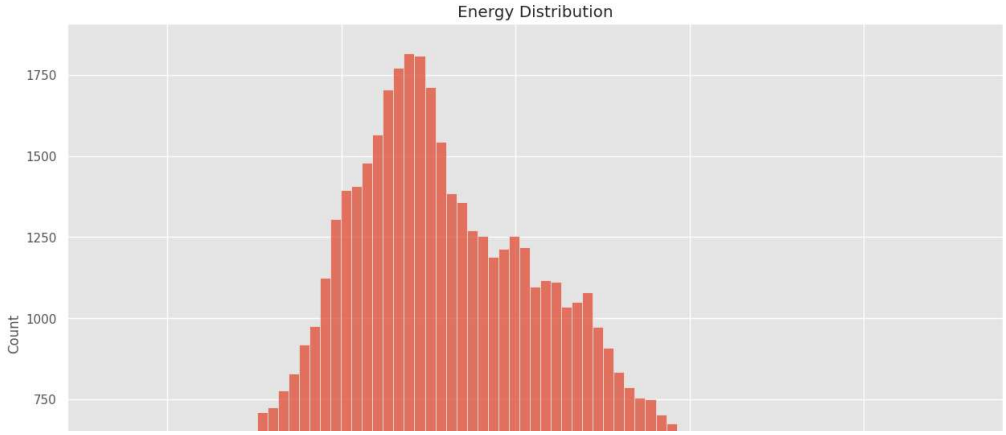


Energy Consumption



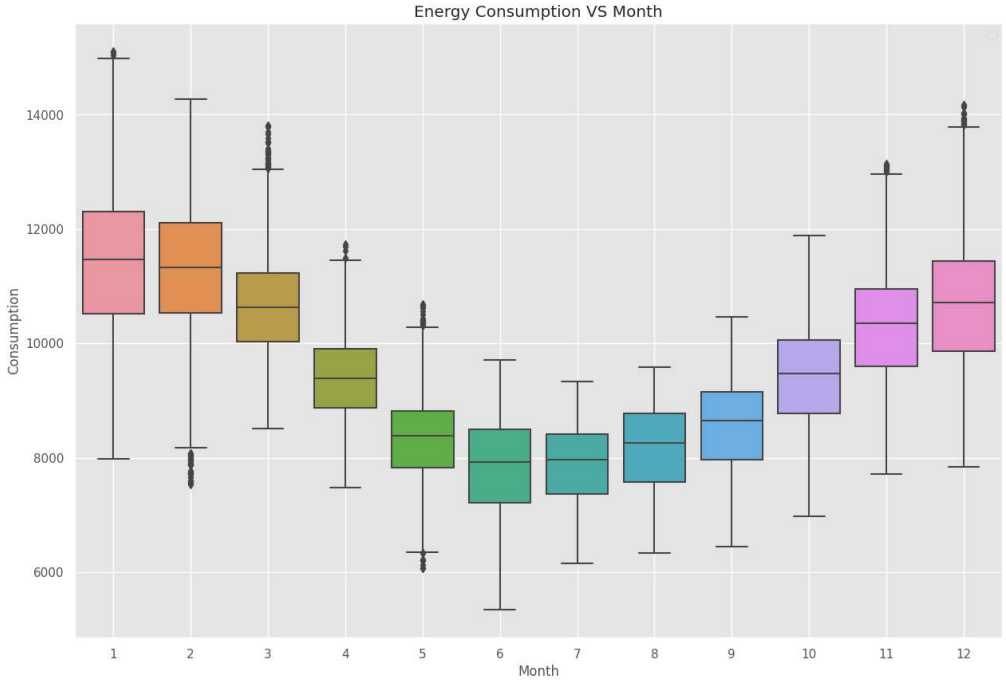

```
#Distribution off Energy Consumption
fig = plt.figure(figsize = (15,10))
sns.histplot(dataset["Consumption"])
plt.title("Energy Distribution")
```

```
Text(0.5, 1.0, 'Energy Distribution')
```




```
fig = plt.figure(figsize = (15,10))
sns.boxplot(x=dataset["Month"], y=dataset["Consumption"], data= df)
plt.title("Energy Consumption VS Month")
plt.xlabel("Month")
plt.grid(True, alpha=1)
plt.legend()

for label in ax1.xaxis.get_ticklabels():
    label.set_rotation(90)
```

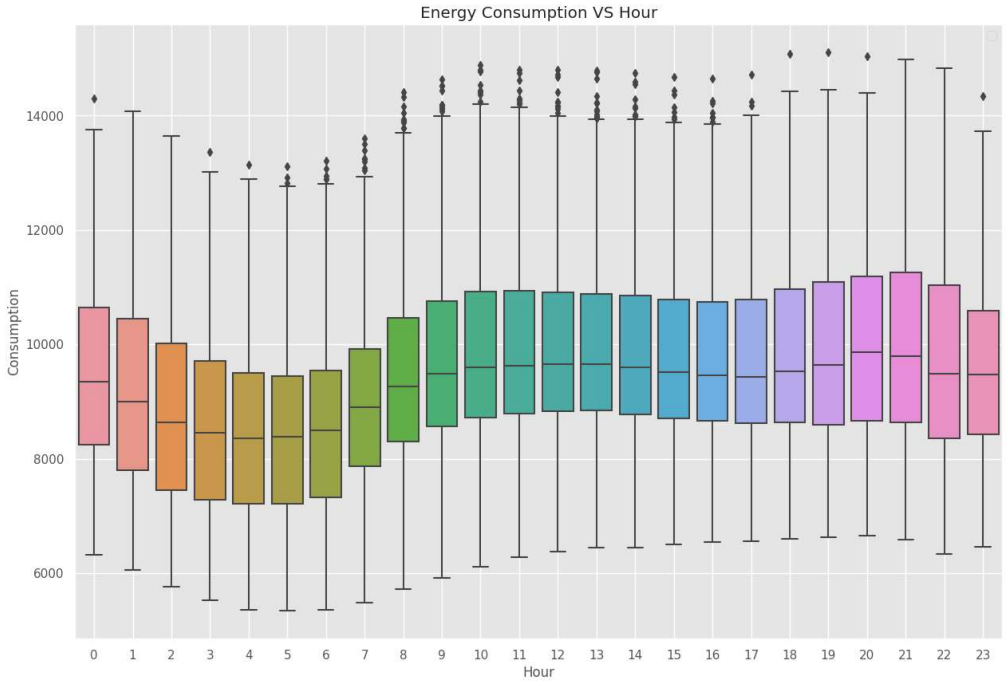
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that art
```




```
dataset1 = dataset
fig = plt.figure(figsize = (15,10))
```

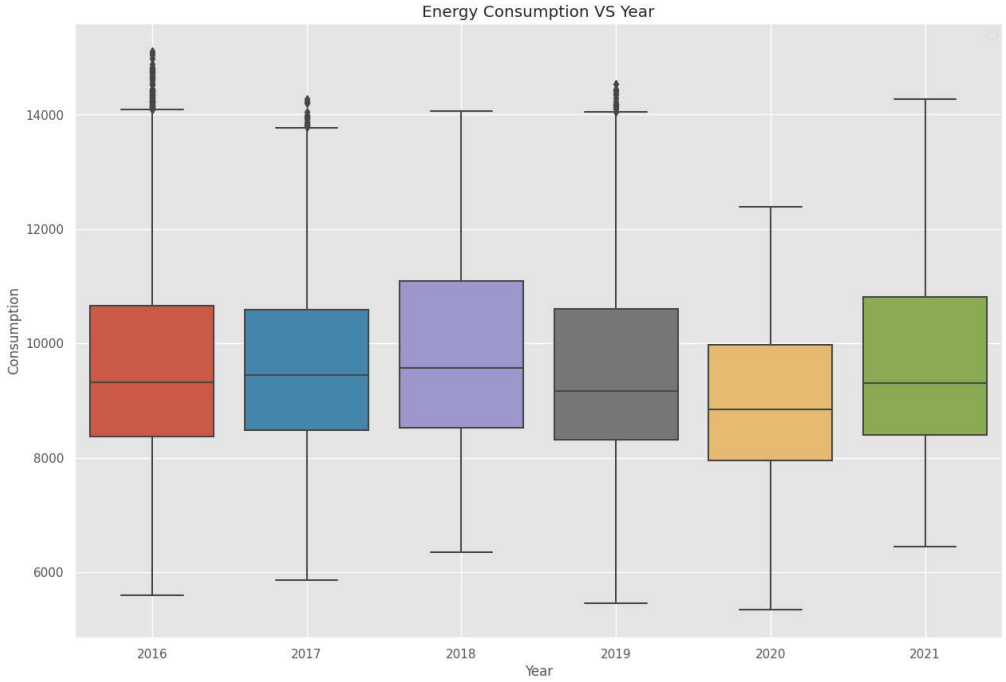
```
sns.boxplot(x=dataset1.index.hour, y=dataset1.Consumption, data= df)
plt.title("Energy Consumption VS Hour")
plt.xlabel("Hour")
plt.grid(True, alpha=1)
plt.legend()

for label in ax1.xaxis.get_ticklabels():
    label.set_rotation(90)
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that art
```




```
fig = plt.figure(figsize = (15,10))
sns.boxplot(x=dataset1.index.year, y=dataset1.Consumption, data= df)
plt.title("Energy Consumption VS Year")
plt.xlabel("Year")
plt.grid(True, alpha=1)
plt.legend()

for label in ax1.xaxis.get_ticklabels():
    label.set_rotation(90)
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that art
```



LSTM Model

Train, Validation and Test Dataset

```
# Downsampling involves decreasing the time-frequency of the data
# Downsapling the time-frequency from hours to days
newDataSet = dataset.resample("D").mean()
```

```
<ipython-input-21-9b075e1ce5f9>:3: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprec
  newDataSet = dataset.resample("D").mean()
```

```
# We have 2193 row
# 2193 - 3 - 6 = 2184 row after omit first two rows and last six ones.
# 2184 / 7 = 312 week
# 312 * 80 %  250 week for train (1750 day)
# 312 - 250 = 62 week for test (434 day)
print("Old Dataset: ", dataset.shape)
print("New Dataset: ", newDataSet.shape)
```

```
Old Dataset:  (52774, 7)
New Dataset:  (2184, 4)
```

```
newDataSet.head()
```

	Consumption	Month	Year	Week
2016-01-07	14243.541667	1.0	2016.0	1.0

```
y = newDataSet["Consumption"]
print(y[0])
y.shape
```

```
12300.625
(2184,)
```

```
# Normalize data before model fitting
# it will boost the performance( in neural networks) + transform
from sklearn.preprocessing import MinMaxScaler
# scale of the output and input inthe range 0-1 to match the scale of the layer of LSTM
scaler = MinMaxScaler(feature_range = (0,1))
# reshape: convert the univariate 1D array into 2D
y = scaler.fit_transform(np.array(y).reshape(-1,1))
print("Normalizing data before model fitting")
print(y[:10])
```

```
Normalizing data before model fitting
[[0.75916744]
 [0.83908687]
 [0.86975003]
 [1.        ]
 [0.98489309]
 [0.82500258]
 [0.78623593]
 [0.83189672]
 [0.80348621]
 [0.8313604 ]]
```

```
training_size = int(len(y)*0.80)
test_size = len(y)- training_size
val_size = int(training_size*0.20)
train_data , test_data , val_data = y[0:training_size-val_size,:] , y[training_size:len(y),:1], y[len(y)-test_size-val_size:
```

```
# building input variable
def create_dataset(dataset, time_step = 1):
  dataX, dataY = [] , []
  for i in range(len(dataset)-time_step-1):
    a = dataset[i:(i+time_step),0]
    dataX.append(a)
    dataY.append(dataset[i + time_step,0])
  return np.array(dataX), np.array(dataY)
```

```
time_step = 100
X_train, y_train = create_dataset(train_data, time_step)
X_test, ytest = create_dataset(test_data, time_step)
X_val, yval = create_dataset(val_data, time_step)
```

```
# reshape train and input-output pairs
X_train = X_train.reshape(X_train.shape[0],X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1],1)
X_val = X_val.reshape(X_val.shape[0], X_val.shape[1],1)

print("X_train shape: ", X_train.shape)
print("X_test shape: ",X_test.shape)
print("X_val shape: ",X_val.shape)
```

```
X_train shape:  (1297, 100, 1)
X_test shape:  (336, 100, 1)
X_val shape:  (248, 100, 1)
```

Model Structure

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

model = Sequential()

# Adding the first LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True, input_shape = (time_step, 1)))
model.add(Dropout(0.2))

# Adding a second LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
# model.add(Dropout(0.2))

# # Adding a third LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50, return_sequences = True))
# model.add(Dropout(0.2))

# Adding a fourth LSTM layer and some Dropout regularisation
model.add(LSTM(units = 50))
# model.add(Dropout(0.2))

# Adding the output layer
model.add(Dense(units = 1))

# Compiling the RNN
model.compile(optimizer = 'adam', loss = 'mean_squared_error')


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 50)           10400

 dropout (Dropout)           (None, 100, 50)           0

 lstm_1 (LSTM)               (None, 100, 50)           20200

 lstm_2 (LSTM)               (None, 100, 50)           20200

 lstm_3 (LSTM)               (None, 50)                20200

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
_________________________________________________________________
```

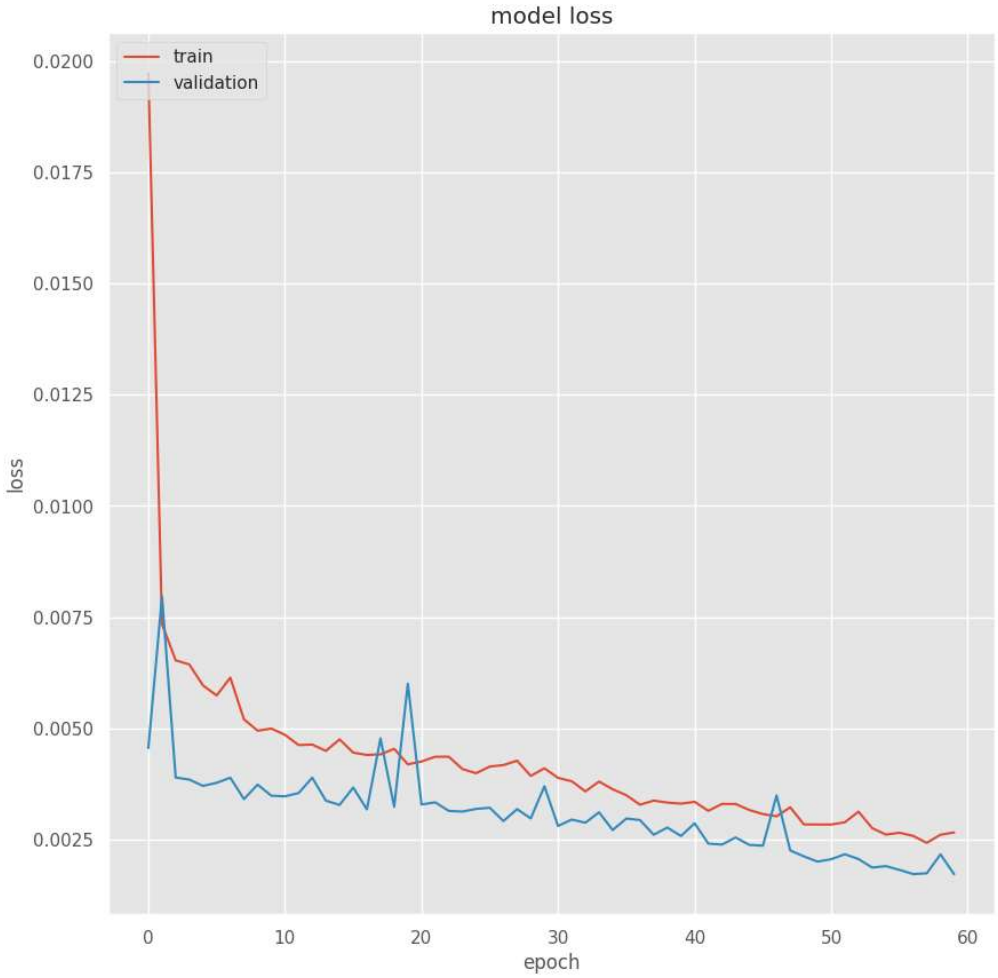
Model Training

```
history = model.fit(X_train, y_train, validation_data = (X_val,yval), verbose = 1,epochs = 60 ,batch_size = 20)
```

```
Epoch 32/60
65/65 [==============================] - 15s 236ms/step - loss: 0.0038 - val_loss: 0.0029
Epoch 33/60
65/65 [==============================] - 15s 234ms/step - loss: 0.0036 - val_loss: 0.0029
Epoch 34/60
65/65 [==============================] - 14s 222ms/step - loss: 0.0038 - val_loss: 0.0031
Epoch 35/60
65/65 [==============================] - 15s 229ms/step - loss: 0.0036 - val_loss: 0.0027
Epoch 36/60
65/65 [==============================] - 15s 224ms/step - loss: 0.0035 - val_loss: 0.0030
Epoch 37/60
65/65 [==============================] - 15s 224ms/step - loss: 0.0033 - val_loss: 0.0029
Epoch 38/60
65/65 [==============================] - 15s 229ms/step - loss: 0.0034 - val_loss: 0.0026
Epoch 39/60
65/65 [==============================] - 15s 236ms/step - loss: 0.0033 - val_loss: 0.0028
Epoch 40/60
65/65 [==============================] - 14s 222ms/step - loss: 0.0033 - val_loss: 0.0026
Epoch 41/60
65/65 [==============================] - 14s 220ms/step - loss: 0.0033 - val_loss: 0.0029
Epoch 42/60
65/65 [==============================] - 14s 221ms/step - loss: 0.0031 - val_loss: 0.0024
Epoch 43/60
65/65 [==============================] - 14s 221ms/step - loss: 0.0033 - val_loss: 0.0024
Epoch 44/60
65/65 [==============================] - 14s 222ms/step - loss: 0.0033 - val_loss: 0.0025
Epoch 45/60
65/65 [==============================] - 16s 243ms/step - loss: 0.0032 - val_loss: 0.0024
Epoch 46/60
65/65 [==============================] - 14s 221ms/step - loss: 0.0031 - val_loss: 0.0024
Epoch 47/60
65/65 [==============================] - 15s 227ms/step - loss: 0.0030 - val_loss: 0.0035
Epoch 48/60
65/65 [==============================] - 15s 233ms/step - loss: 0.0032 - val_loss: 0.0022
Epoch 49/60
65/65 [==============================] - 14s 219ms/step - loss: 0.0028 - val_loss: 0.0021
Epoch 50/60
65/65 [==============================] - 14s 221ms/step - loss: 0.0028 - val_loss: 0.0020
Epoch 51/60
65/65 [==============================] - 14s 220ms/step - loss: 0.0028 - val_loss: 0.0021
Epoch 52/60
65/65 [==============================] - 14s 221ms/step - loss: 0.0029 - val_loss: 0.0022
Epoch 53/60
65/65 [==============================] - 14s 220ms/step - loss: 0.0031 - val_loss: 0.0021
Epoch 54/60
65/65 [==============================] - 14s 221ms/step - loss: 0.0028 - val_loss: 0.0019
Epoch 55/60
65/65 [==============================] - 14s 218ms/step - loss: 0.0026 - val_loss: 0.0019
Epoch 56/60
65/65 [==============================] - 14s 220ms/step - loss: 0.0026 - val_loss: 0.0018
Epoch 57/60
65/65 [==============================] - 17s 261ms/step - loss: 0.0026 - val_loss: 0.0017
Epoch 58/60
65/65 [==============================] - 14s 221ms/step - loss: 0.0024 - val_loss: 0.0017
Epoch 59/60
65/65 [==============================] - 14s 220ms/step - loss: 0.0026 - val_loss: 0.0022
Epoch 60/60
65/65 [==============================] - 14s 222ms/step - loss: 0.0027 - val_loss: 0.0017
```

Model Evaluating

```
plt.figure(figsize=(10,10))
plt.plot(history.history['loss']) # tb
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```




```
import tensorflow as tf
tf.__version__
```

```
'2.12.0'
```

```
### prediction and check performance metrics
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
val_predict=model.predict(X_val)
```

```
41/41 [==============================] - 4s 61ms/step
11/11 [==============================] - 1s 63ms/step
8/8 [==============================] - 1s 65ms/step
```

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

```
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

```
9544.419392049154
```

```
print(train_predict.shape)
print(test_predict.shape)
print(val_predict.shape)
print(train_predict[0])
print(y_train.shape)
```

```
(1297, 1)
(336, 1)
(248, 1)
[9315.555]
(1297,)
```

Conclusion

```
# Predicting consumption using training data
train_predictions = model.predict(X_train)
train_predictions =scaler.inverse_transform(train_predictions)

y_train = y_train.reshape(y_train.shape[0], 1)
actual = scaler.inverse_transform(y_train)
train_results = pd.DataFrame()

train_results["Train Predictions"] = train_predictions.tolist()
train_results["Actuals"] = actual.tolist()

train_results
```

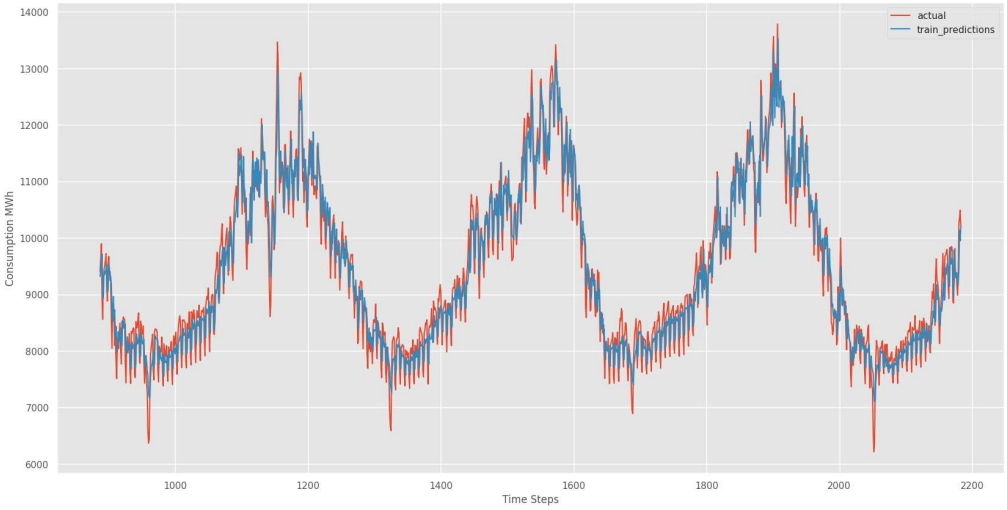
```
41/41 [==============================] - 2s 60ms/step
```

	Train Predictions	Actuals
0	[9315.5546875]	[9406.708333333334]
1	[9456.650390625]	[9614.791666666666]
2	[9535.1259765625]	[9894.708333333334]
3	[9713.736328125]	[8933.708333333334]
4	[8923.240234375]	[8557.208333333334]
...	...	...
1292	[9163.5908203125]	[9259.666666666666]
1293	[9411.6298828125]	[10248.5]
1294	[10133.6025390625]	[10360.333333333334]
1295	[9947.0087890625]	[10489.833333333334]
1296	[10198.3828125]	[10204.5]

1297 rows × 2 columns

```
plt.figure(figsize=(20,10))

# Actual Consumption
ActualPlot = np.empty_like(y )
ActualPlot[:, :] = np.nan
ActualPlot[887:, :] = actual.tolist()
```

```
plt.plot(ActualPlot)

TrainPredictionsPlot = np.empty_like(y )
TrainPredictionsPlot[:, :] = np.nan
TrainPredictionsPlot[887:, :] = train_predictions.tolist()
plt.plot(TrainPredictionsPlot)

plt.legend(['actual','train_predictions'])
plt.xlabel('Time Steps')
plt.ylabel('Consumption MWh')
plt.show()
```




```
# Predicting consumption using validation data
val_predictions = model.predict(X_val)
val_predictions =scaler.inverse_transform(val_predictions)

yval = yval.reshape(yval.shape[0], 1)
actual_val = scaler.inverse_transform(yval)

val_results = pd.DataFrame()
val_results["Val Predictions"] = val_predictions.tolist()
val_results["Actuals_val"] = actual_val.tolist()

val_results
```

```
8/8 [==============================] - 0s 58ms/step
```

	Val Predictions	Actuals_val
0	[9137.9443359375]	[8931.875]
1	[9522.9443359375]	[9112.916666666666]
2	[9444.837890625]	[10019.416666666666]
3	[10092.984375]	[10290.375]

```
plt.figure(figsize=(20,10))

# Actual Consumption
ActualPlot = np.empty_like(y)
ActualPlot[:, :] = np.nan
ActualPlot[1936:, :] = actual_val.tolist()
plt.plot(ActualPlot)

# Validation Prediction
ValPredictionsPlot = np.empty_like(y)
ValPredictionsPlot[:, :] = np.nan
ValPredictionsPlot[1936:, :] = val_predictions.tolist()
plt.plot(ValPredictionsPlot)

plt.legend(['actual_val','Val_predictions'])
plt.xlabel('Time Steps')
plt.ylabel('Consumption MWh')
```

```
Text(0, 0.5, 'Consumption MWh')
```


```
# Predicting consumption using test data
test_predictions = model.predict(X_test)
test_predictions =scaler.inverse_transform(test_predictions)

ytest = ytest.reshape(ytest.shape[0], 1)
actual_test = scaler.inverse_transform(ytest)

test_results = pd.DataFrame()
test_results["test Predictions"] = test_predictions.tolist()
test_results["Actuals_test"] = actual_test.tolist()
```

```
test_results
```

```
11/11 [==============================] - 1s 57ms/step
```

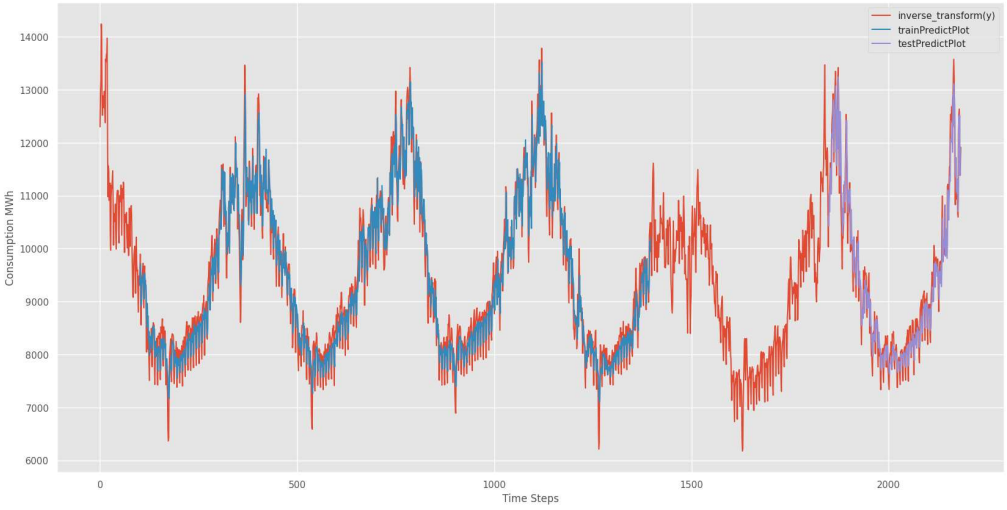
	test Predictions	Actuals_test
0	[10607.3056640625]	[10032.541666666666]
1	[10424.5283203125]	[10732.125]
2	[11035.681640625]	[10733.583333333334]
3	[10693.8857421875]	[10971.875]
4	[11040.0947265625]	[11227.791666666666]
...	...	...
331	[12497.392578125]	[12540.25]
332	[12141.0068359375]	[12635.958333333334]
333	[12520.8193359375]	[11684.333333333334]
334	[11394.19921875]	[11384.166666666666]
335	[11905.556640625]	[11581.625]

336 rows × 2 columns

```
plt.figure(figsize=(20,10))

# Actual Consumption
ActualPlot = np.empty_like(y)
ActualPlot[:, :] = np.nan
ActualPlot[1848:, :] = actual_test.tolist()
plt.plot(ActualPlot)

# Test Prediction
TestPredictionsPlot = np.empty_like(y)
TestPredictionsPlot[:, :] = np.nan
TestPredictionsPlot[1848:, :] = test_predictions.tolist()
plt.plot(TestPredictionsPlot)

plt.legend(['Actual_test','Test_predictions'])
plt.xlabel('Time Steps')
plt.ylabel('Consumption MWh')
```

```
Text(0, 0.5, 'Consumption MWh')
```


```
### Plotting
# shift train predictions for plotting
look_back=100 #****************************************>>>>>100
fig, ax = plt.subplots(figsize=(20,10))
trainPredictPlot = np.empty_like(y)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = np.empty_like(y)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1+349:len(y)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(y))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.legend(['inverse_transform(y)','trainPredictPlot','testPredictPlot'])
plt.xlabel('Time Steps')
plt.ylabel('Consumption MWh')
plt.show()
```



Future forecasting

```
print(len(test_data))
print(test_data[0])
print(len(train_data))
x_input=test_data[337:].reshape(1,-1)
print(x_input.shape)
# print(x_input[0])
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
# print(temp_input)
```

```
437
[0.40285095]
1398
(1, 100)
```

```
# demonstrate prediction for next 30 days
from numpy import array

lst_output=[]
n_steps=100
i=0
test = ""
while(i<30):

    if(len(temp_input)>100):
        test = "if"
        #print(temp_input)
        x_input=np.array(temp_input[1:])
        print("{} day input {}".format(i,x_input))
        x_input=x_input.reshape(1,-1)
        x_input = x_input.reshape((1, n_steps, 1))
        #print(x_input)
        yhat = model.predict(x_input, verbose=0)
        print("{} day output {}".format(i,yhat))
        temp_input.extend(yhat[0].tolist())
        temp_input=temp_input[1:]
        #print(temp_input)
        lst_output.extend(yhat.tolist())
        i=i+1
    else:
        test="else"
        x_input = x_input.reshape((1, n_steps,1))
        yhat = model.predict(x_input, verbose=0)
        #print(yhat[0])
        temp_input.extend(yhat[0].tolist())
        #print(len(temp_input))
        lst_output.extend(yhat.tolist())
        i=i+1


print(test)
print(len(lst_output))
```

```
 0.46884619 0.620282   0.62459457 0.57526598 0.59118893 0.62118583
 0.6655046  0.66726578 0.74881727 0.7883793  0.77671212 0.78571429
 0.80647144 0.74484041 0.82660366 0.82588059 0.87828737 0.91723995
 0.87967669 0.80140998 0.65632683 0.57681025 0.63147402 0.59401921
 0.59759839 0.5892883  0.61484351 0.55500981 0.54768619 0.74439624
 0.78886995 0.8007334  0.68277554 0.64556864 0.67004442 0.71099576
 0.75049233 0.76867157 0.77195299 0.77108955 0.77085268 0.77211601
 0.77416223 0.77608794 0.77738833 0.77795547 0.77787334 0.77732879
 0.77650958 0.77553916 0.77448195 0.77338749 0.77227324 0.77111286
 0.76989353 0.76862258 0.76730561 0.76593935]
22 day output [[0.764537]]
23 day input [0.33202665 0.36575767 0.38037909 0.41900114 0.40965809 0.36264332
 0.34998967 0.43912819 0.48126743 0.46479186 0.40083153 0.44962814
 0.44524842 0.45453466 0.45160624 0.42284888 0.4359467  0.44960748
 0.42327755 0.3690941  0.3264797  0.41017457 0.42818407 0.44249561
 0.41147609 0.44483008 0.39457184 0.43183555 0.53842578 0.59736597
 0.49454602 0.47844231 0.49448921 0.47407293 0.45992666 0.57367524
 0.49652928 0.4854922  0.46876872 0.51486933 0.46267431 0.46884619
 0.620282   0.62459457 0.57526598 0.59118893 0.62118583 0.6655046
 0.66726578 0.74881727 0.7883793  0.77671212 0.78571429 0.80647144
 0.74484041 0.82660366 0.82588059 0.87828737 0.91723995 0.87967669
 0.80140998 0.65632683 0.57681025 0.63147402 0.59401921 0.59759839
 0.5892883  0.61484351 0.55500981 0.54768619 0.74439624 0.78886995
 0.8007334  0.68277554 0.64556864 0.67004442 0.71099576 0.75049233
 0.76867157 0.77195299 0.77108955 0.77085268 0.77211601 0.77416223
 0.77608794 0.77738833 0.77795547 0.77787334 0.77732879 0.77650958
 0.77553916 0.77448195 0.77338749 0.77227324 0.77111286 0.76989353
 0.76862258 0.76730561 0.76593935 0.76453698]
23 day output [[0.7631034]]
24 day input [0.36575767 0.38037909 0.41900114 0.40965809 0.36264332 0.34998967
 0.43912819 0.48126743 0.46479186 0.40083153 0.44962814 0.44524842
 0.45453466 0.45160624 0.42284888 0.4359467  0.44960748 0.42327755
```

```
print(len(y))
day_new=np.arange(1,101)
day_pred=np.arange(101, 131 )
plt.figure(figsize = (15,10))
plt.plot(day_new,scaler.inverse_transform(y[2084:]))
plt.plot(day_pred,scaler.inverse_transform(lst_output))
print(scaler.inverse_transform(lst_output))
```

```
2184
[[12230.63857416]
 [12377.29957546]
 [12403.77238701]
 [12396.80664221]
 [12394.8957027 ]
 [12405.0875404 ]
 [12421.59548039]
 [12437.13112051]
 [12447.62205342]
 [12452.19744081]
 [12451.53481508]
 [12447.14167381]
 [12440.53272749]
 [12432.70383815]
 [12424.17481596]
 [12415.34525598]
 [12406.35605032]
 [12396.99465866]
 [12387.15769599]
 [12376.90430815]
 [12366.27969603]
 [12355.25741229]
 [12343.94372712]
 [12332.37855191]
 [12320.402241  ]
 [12307.95035909]
 [12295.07435823]
 [12281.79539631]
 [12268.19425787]
 [12254.4108733 ]]
```


```
df3=y.tolist()
df3.extend(lst_output)
plt.figure(figsize=(15,10))
plt.plot(df3[2000:])
```