Wind Energy Prediction and Analysis

Importing Basic Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import datetime
import time

%matplotlib inline
sns.set(rc = {"figure.figsize" : (8, 6)})
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[1], line 3
      1 import pandas as pd
      2 import numpy as np
----> 3 import matplotlib.pyplot as plt
      4 import seaborn as sns
      6 import datetime

ModuleNotFoundError: No module named 'matplotlib'
```

Reading the dataset

In [2]:
```
data = pd.read_csv("../input/wind-turbine-scada-dataset/T1.csv")
data.head()
```

Out[2]:

	Date/Time	LV ActivePower (kW)	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)
0	01 01 2018 00:00	380.047791	5.311336	416.328908	259.994904
1	01 01 2018 00:10	453.769196	5.672167	519.917511	268.641113
2	01 01 2018 00:20	306.376587	5.216037	390.900016	272.564789
3	01 01 2018 00:30	419.645905	5.659674	516.127569	271.258087
4	01 01 2018 00:40	380.650696	5.577941	491.702972	265.674286

Data Wrangling

In [3]:
```
data.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50530 entries, 0 to 50529
Data columns (total 5 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Date/Time                      50530 non-null  object
 1   LV ActivePower (kW)            50530 non-null  float64
 2   Wind Speed (m/s)               50530 non-null  float64
 3   Theoretical_Power_Curve (KWh)  50530 non-null  float64
 4   Wind Direction (°)             50530 non-null  float64
dtypes: float64(4), object(1)
memory usage: 1.9+ MB
```

In [4]:
```
data.describe()
```

Out[4]:

	LV ActivePower (kW)	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)
count	50530.000000	50530.000000	50530.000000	50530.000000
mean	1307.684332	7.557952	1492.175463	123.687559
std	1312.459242	4.227166	1368.018238	93.443736
min	-2.471405	0.000000	0.000000	0.000000
25%	50.677890	4.201395	161.328167	49.315437
50%	825.838074	7.104594	1063.776283	73.712978
75%	2482.507568	10.300020	2964.972462	201.696720
max	3618.732910	25.206011	3600.000000	359.997589

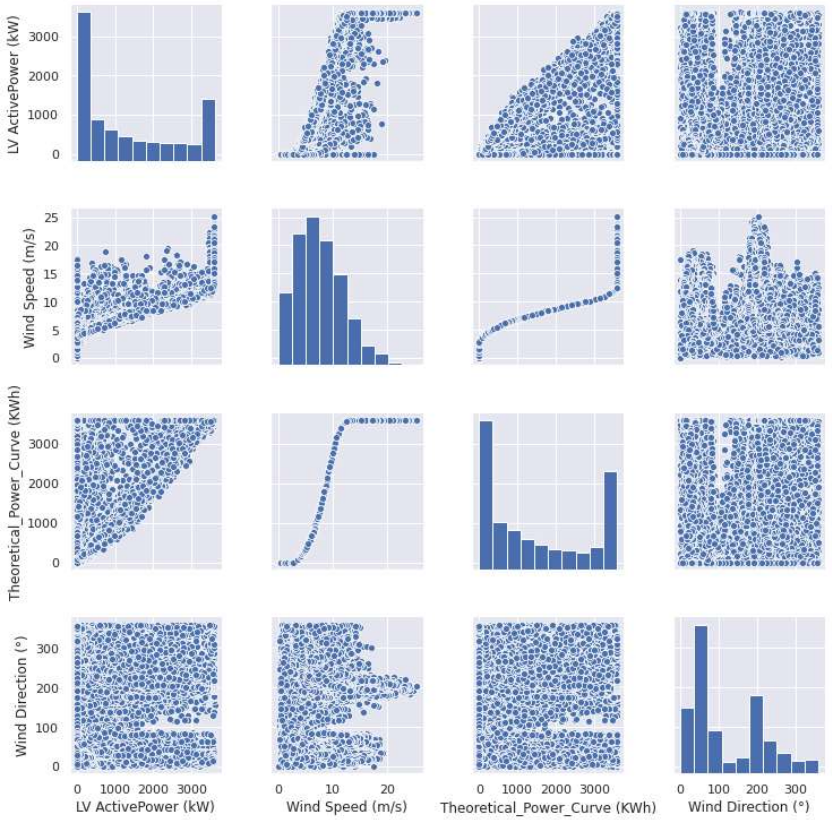
In [5]:
```
data.isnull().sum()
```

Out[5]:
```
Date/Time                      0
LV ActivePower (kW)            0
Wind Speed (m/s)               0
Theoretical_Power_Curve (KWh)  0
Wind Direction (°)             0
dtype: int64
```

Visualization

In [6]:
```
# Pair Plot correlation between all attributes
sns.pairplot(data)
```

Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x7fd743d1ce90>
```



In [7]:
```
#correlation between the values
corr = data.corr()
plt.figure(figsize=(10, 8))

ax = sns.heatmap(corr, vmin = -1, vmax = 1, annot = True)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
plt.show()
corr
```


Out[7]:

	LV ActivePower (kW)	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wi Directi
LV ActivePower (kW)	1.000000	0.912774	0.949918	-0.0627
Wind Speed (m/s)	0.912774	1.000000	0.944209	-0.0771
Theoretical_Power_Curve (KWh)	0.949918	0.944209	1.000000	-0.0990
Wind Direction (°)	-0.062702	-0.077188	-0.099076	1.0000

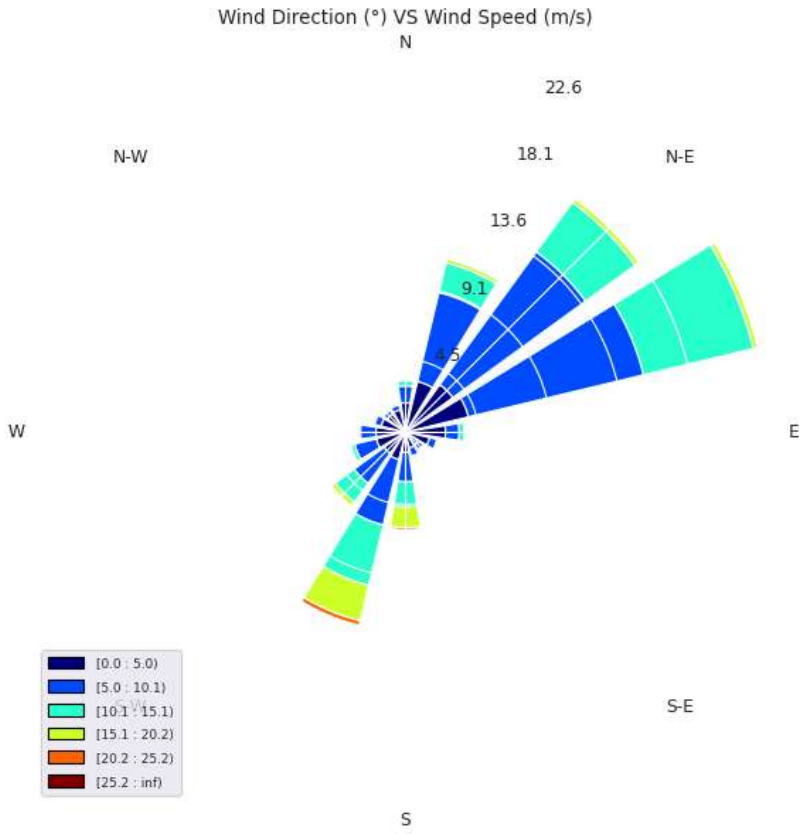
In [8]:
```
# Importing a visualization library
! pip install windrose
```

```
Collecting windrose
  Downloading windrose-1.6.7-py2.py3-none-any.whl (20 kB)
Requirement already satisfied: matplotlib in /opt/conda/lib/python3.7/site-packag
es (from windrose) (3.2.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (f
rom windrose) (1.18.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /opt/c
onda/lib/python3.7/site-packages (from matplotlib->windrose) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site
-packages (from matplotlib->windrose) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-pack
ages (from matplotlib->windrose) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /opt/conda/lib/python3.7/s
ite-packages (from matplotlib->windrose) (2.8.1)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (fro
m cycler>=0.10->matplotlib->windrose) (1.14.0)
Installing collected packages: windrose
Successfully installed windrose-1.6.7
```

Pie Bar Chart (Wind Direction VS Wind Speed)

In [9]:
```
from windrose import WindroseAxes

ax = WindroseAxes.from_ax()
ax.bar(data['Wind Direction (°)'], data['Wind Speed (m/s)'], normed=True, openin
ax.set_legend()
plt.title("Wind Direction (°) VS Wind Speed (m/s)")
plt.show()
```

```
/opt/conda/lib/python3.7/site-packages/windrose/windrose.py:29: MatplotlibDepreca
tionWarning:
The Appender class was deprecated in Matplotlib 3.1 and will be removed in 3.3.
  addendum = docstring.Appender(msg, "\n\n")
/opt/conda/lib/python3.7/site-packages/windrose/windrose.py:30: MatplotlibDepreca
tionWarning:
The copy_dedent function was deprecated in Matplotlib 3.1 and will be removed in
3.3. Use docstring.copy() and cbook.dedent() instead.
  return lambda func: addendum(docstring.copy_dedent(base)(func))
/opt/conda/lib/python3.7/site-packages/windrose/windrose.py:30: MatplotlibDepreca
tionWarning:
The dedent function was deprecated in Matplotlib 3.1 and will be removed in 3.3.
Use inspect.getdoc() instead.
  return lambda func: addendum(docstring.copy_dedent(base)(func))
/opt/conda/lib/python3.7/site-packages/windrose/windrose.py:30: MatplotlibDepreca
tionWarning:
The dedent function was deprecated in Matplotlib 3.1 and will be removed in 3.3.
Use inspect.cleandoc instead.
  return lambda func: addendum(docstring.copy_dedent(base)(func))
```

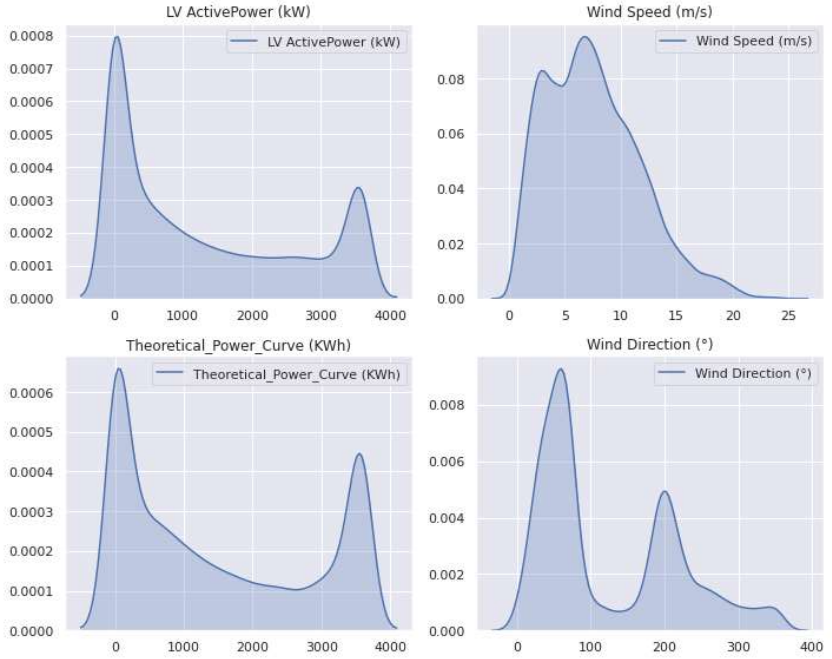
In [10]:
```
# Spliting the date time in year, month, days, hours and minutes
data['Year']=data['Date/Time'].apply(lambda x: time.strptime(x,"%d %m %Y %H:%M")
data['Month']=data['Date/Time'].apply(lambda x: time.strptime(x,"%d %m %Y %H:%M"
data['Day']=data['Date/Time'].apply(lambda x: time.strptime(x,"%d %m %Y %H:%M")[
data['Time_Hours']=data['Date/Time'].apply(lambda x: time.strptime(x,"%d %m %Y %
data['Time_Minutes']=data['Date/Time'].apply(lambda x: time.strptime(x,"%d %m %Y
data.head(10)
```

Out[10]:

	Date/Time	LV ActivePower (kW)	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)	Year
0	01 01 2018 00:00	380.047791	5.311336	416.328908	259.994904	2018
1	01 01 2018 00:10	453.769196	5.672167	519.917511	268.641113	2018
2	01 01 2018 00:20	306.376587	5.216037	390.900016	272.564789	2018
3	01 01 2018 00:30	419.645905	5.659674	516.127569	271.258087	2018
4	01 01 2018 00:40	380.650696	5.577941	491.702972	265.674286	2018
5	01 01 2018 00:50	402.391998	5.604052	499.436385	264.578613	2018
6	01 01 2018 01:00	447.605713	5.793008	557.372363	266.163605	2018
7	01 01 2018 01:10	387.242188	5.306050	414.898179	257.949493	2018
8	01 01 2018 01:20	463.651215	5.584629	493.677652	253.480698	2018
9	01 01 2018 01:30	439.725708	5.523228	475.706783	258.723785	2018

KDE Plot

In [11]:
```
#plotting the data distribution
plt.figure(figsize=(10, 8))
for i in range(4):
    plt.subplot(2, 2, i+1)
    sns.kdeplot(data.iloc[:,i+1], shade=True)
    plt.title(data.columns[i+1])
plt.tight_layout()
plt.show()
```

In [12]:
```
# droping all the null values from the data
data = data.dropna()
```

Converting the Data/Time feature in proper DateTime format

In [13]:
```
data["Date/Time"] = pd.to_datetime(data["Date/Time"], format = "%d %m %Y %H:%M",
data
```

Out[13]:

	Date/Time	LV ActivePower (kW)	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)	Wind Direction (°)
0	2018-01-01 00:00:00	380.047791	5.311336	416.328908	259.994904
1	2018-01-01 00:10:00	453.769196	5.672167	519.917511	268.641113
2	2018-01-01 00:20:00	306.376587	5.216037	390.900016	272.564789
3	2018-01-01 00:30:00	419.645905	5.659674	516.127569	271.258087
4	2018-01-01 00:40:00	380.650696	5.577941	491.702972	265.674286
...	...	...	...	...	...
50525	2018-12-31 23:10:00	2963.980957	11.404030	3397.190793	80.502724
50526	2018-12-31 23:20:00	1684.353027	7.332648	1173.055771	84.062599
50527	2018-12-31 23:30:00	2201.106934	8.435358	1788.284755	84.742500
50528	2018-12-31 23:40:00	2515.694092	9.421366	2418.382503	84.297913
50529	2018-12-31 23:50:00	2820.466064	9.979332	2779.184096	82.274620

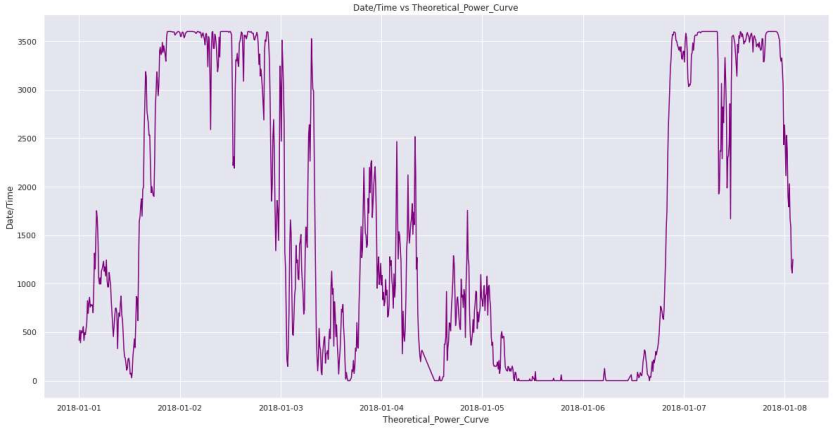
50530 rows × 10 columns

Line Graph of DateTime VS Target variable

In [14]:
```
# Create figure and plot space
fig, ax = plt.subplots(figsize=(20,10))

# Add x-axis and y-axis
ax.plot(data['Date/Time'][0:1000],
        data['Theoretical_Power_Curve (KWh)'][0:1000],
        color='purple')

# Set title and labels for axes
ax.set(xlabel="Theoretical_Power_Curve",
```



```
       ylabel="Date/Time",
       title="Date/Time vs Theoretical_Power_Curve")

plt.show()
```



In [15]:
```
# Droping all the irrelavent features that dosent affect the target variable.
cols = ["Wind Direction (°)", 'Year', 'Month', "Day", "Time_Hours", "Time_Minute
data = data.drop(cols, axis=1)
data.head()
```

Out[15]:

	Date/Time	LV ActivePower (kW)	Wind Speed (m/s)	Theoretical_Power_Curve (KWh)
0	2018-01-01 00:00:00	380.047791	5.311336	416.328908
1	2018-01-01 00:10:00	453.769196	5.672167	519.917511
2	2018-01-01 00:20:00	306.376587	5.216037	390.900016
3	2018-01-01 00:30:00	419.645905	5.659674	516.127569
4	2018-01-01 00:40:00	380.650696	5.577941	491.702972

In [16]:
```
df = data.copy()
```

Building the LSTM model

In [17]:
```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy as np
import pandas as pd
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:5: FutureWarning: Th
e pandas.datetime class is deprecated and will be removed from pandas in a future
version. Import from datetime module instead.
  """
Using TensorFlow backend.
```

In [18]:
```
# converting the Data/Time as the index for proper shape of the input.
df = df.set_index('Date/Time')
```

In [19]:
```
# Hardcode all variables
batch_size_exp = 1
epoch_exp = 15
neurons_exp = 10
predict_values_exp = 1000
lag_exp=24
```

In [20]:
```
# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df
```

In [21]:
```
# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)
```

In [22]:
```
# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]
```

In [23]:
```
# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled
```


In [24]:
```
# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = np.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]
```

In [25]:
```
# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=1, shuffle=Fals
        model.reset_states()
    return model
    print(model.summary)
```

In [26]:
```
# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    #print(X)
    yhat = model.predict(X, batch_size=1)
    return yhat[0,0]
```

In [27]:
```
'''Drop all the features as we will not be having any in production'''
del df['LV ActivePower (kW)']
del df['Wind Speed (m/s)']
df.head()
```

Out[27]:

Date/Time	Theoretical_Power_Curve (KWh)
2018-01-01 00:00:00	416.328908
2018-01-01 00:10:00	519.917511
2018-01-01 00:20:00	390.900016
2018-01-01 00:30:00	516.127569
2018-01-01 00:40:00	491.702972

In [28]:
```
for i in range(0,10):
  df = df[:-1]
df.tail()
```

Out[28]:

Date/Time	Theoretical_Power_Curve (KWh)
2018-12-31 21:30:00	1811.263260
2018-12-31 21:40:00	1787.100338
2018-12-31 21:50:00	2777.504103
2018-12-31 22:00:00	3025.199012
2018-12-31 22:10:00	3161.693967

In [29]:
```
# transform data to be stationary
raw_values = df.values
diff_values = difference(raw_values, 1)
```

In [30]:
```
# transform data to be supervised learning
supervised = timeseries_to_supervised(diff_values, lag_exp)
supervised_values = supervised.values
```

In [31]:
```
# split data into train and test-sets
train, test = supervised_values[0:-predict_values_exp], supervised_values[-predi
```

In [32]:
```
# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)
```

In [33]:
```
# fit the model
lstm_model = fit_lstm(train_scaled, batch_size_exp, epoch_exp, neurons_exp)
```

```
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0057
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 88s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 88s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 86s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 86s 2ms/step - loss: 0.0056
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0055
49519/49519 [==============================] - 86s 2ms/step - loss: 0.0055
49519/49519 [==============================] - 86s 2ms/step - loss: 0.0055
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0055
49519/49519 [==============================] - 86s 2ms/step - loss: 0.0055
49519/49519 [==============================] - 87s 2ms/step - loss: 0.0055
```

In [ ]:
```
# walk-forward validation on the test data
predictions = list()
expectations = list()
predictions_plot = list()
expectations_plot = list()
test_pred = list()
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)#batch_size_exp to 1
    '''# Start Debug prints
    print("X: %", X)
```



```
    print("yhat: %", yhat)
    # End Debug prints'''
    # Replacing value in test scaled with the predicted value.
    test_pred = [yhat] + test_pred
    if len(test_pred) > lag_exp+1:
        test_pred = test_pred[:-1]
    if i+1<len(test_scaled):
        if i+1 > lag_exp+1:
            test_scaled[i+1] = test_pred
        else:
            test_scaled[i+1] = np.concatenate((test_pred, test_scaled[i+1, i+1:]

    # invert scaling
    yhat = invert_scale(scaler, X, yhat)
    # invert differencing
    yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
    # store forecast
    expected = raw_values[len(train) + i + 1]
    predictions_plot.append(yhat)
    expectations_plot.append(expected)
    if expected != 0:
        predictions.append(yhat)
        expectations.append(expected)
    print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))
```

In [ ]:
```
lstm_model.summary()
```

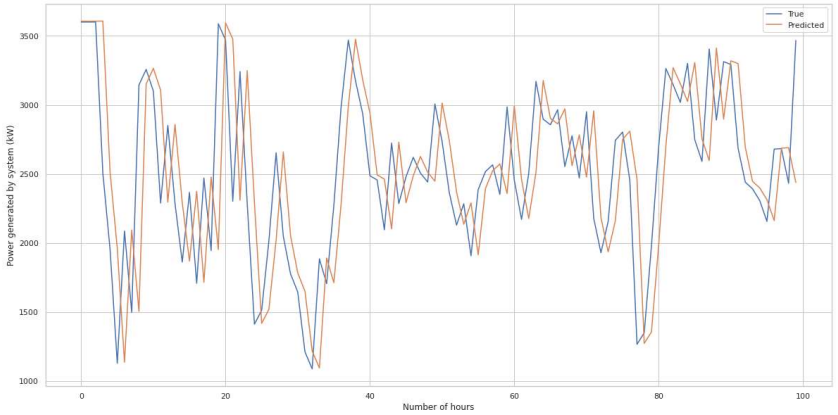
Calculating Mean Absolute Error

In [35]:
```
expectations = np.array(expectations)
predictions = np.array(predictions)
print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations - predictio
```

```
Mean Absolute Percent Error:  0.4614952886022825
```

Final Prediction Plot

In [38]:
```
# line plot of observed vs predicted
sns.set_style("whitegrid")
pyplot.figure(figsize=(20,10))
pyplot.plot(expectations_plot[0:100], label="True")
pyplot.plot(predictions_plot[0:100], label="Predicted")
pyplot.legend(loc='upper right')
pyplot.xlabel("Number of hours")
pyplot.ylabel("Power generated by system (kW)")
pyplot.show()
```


True
Predicted
Power generated by system (kW)
Number of hours
3500
3000
2500
2000
1500
1000
0
20
40
60
80
100