DERİN ÖĞRENME YÖNTEMLERİ İLE ELEKTRİK TÜKETİMİNİN TAHMİNİ

DENİZ CAN TOŞUR

ANTALYA,2021

ÖZET

Bu rapor çalışmasında, 2011-2013 yılları arasında bir bölgenin kullandığı elektrik tüketimini gösteren bir veri seti ve Multivariable(Çok değişkenli) girişli 2006-2011 arasında bir bölgenin kullandığı elektrik tüketimini 'Jupyter Notebook' ile verileri ekrana göstererek daha sonra bu veri setleri ile veri görselleştirmesi, güzelleştirmesi ve düzenlemesi yapılmıştır. ARIMA ve LSTM modellerinde tek girişli veri seti, GRU Modelinde ise Multivariable(Çok değişkenli) veri seti kullanılmıştır. ARIMA ve LSTM aynı model içerisinde GRU ise ayrı model olarak yazıldı. Öncelikle her iki modeldede önce kütüphaneler belirlendi daha sonra veri seti ekrana yansıtılıp veri setini işledikten(düzenledikten) sonra grafikler ile güzelleştirme yapılıp modeller oluşturuldu. Son olarakta ARIMA ve LSTM'nin tahmin modelleri grafiklere dökülüp gerçek veri ile kıyaslatıldı. GRU modelinde ise gerçek değer ile GRU tahmin modeli kıyaslatıldı. Grafik üzerinde gösterme yöntemini kullandığımız fonksiyonlar Çizgi grafiği için 'lineplot', Kutu grafiği için 'boxplot', Histogram grafiği için 'distplot', Violin(Keman) Grafiği için 'violinplot' fonksiyonlarını kullanarak veri seti grafiklere döküldü. Daha sonra veri ön işleme adımları uygulanarak 3 model geliştirildi bu modeller tek değişkenli ARIMA ve LSTM, çok değişkenli GRU modeli. ARIMA ve LSTM modelleri yapıldıktan sonra önce birbirileri ile daha sonra gerçek veri ile 3 veri karşılaştırıldı.

Anahtar Kelimeler: lineplot, boxplot, distplot, violinplot, LSTM,ARIMA,GRU

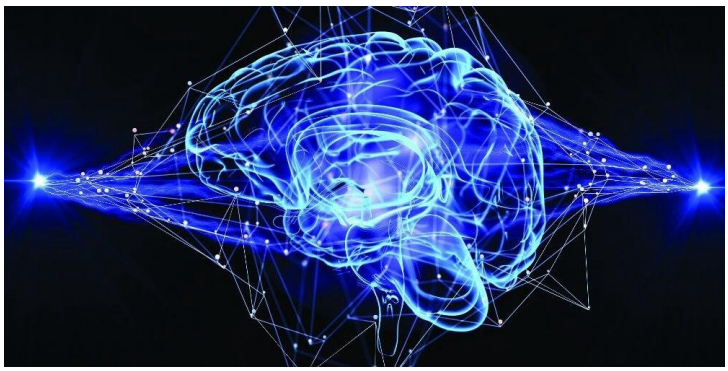
ABSTRACT

In this report, a data set showing the electricity consumption of a data server between 2011-2013 and the electricity consumption carried out between 2006-2011 with a multivariable input was shown in the crotch with the 'Jupyter Notebook', and then the data visualization, beautification and arrangement with this data head. Single-input data set was used in ARIMA and LSTM models and Multivariate (Multivariate) data set was used in GRU Model. ARIMA and LSTM are the same model 'GRU also written as separate models. First of all, libraries were determined in both models, then the models were created after the data set was projected on the screen and after processing (editing) the data set, beautified with graphics. Finally, the estimates of ARIMA and LSTM were graphed and compared with the real data. In the GRU model, the actual value and the estimation of the GRU model were compared. Functions using the method of displaying on the chart The data set was converted to the charts using the functions 'lineplot' for line chart, 'boxplot' for box chart, 'distplot' for histogram chart, 'violin chart' for Violin Chart. Then, by continuing the data pre-processing, 3 models were developed, these models were univariate ARIMA and LSTM, multivariate GRU model. After the ARIMA and LSTM models were made, 3 data were compared first with each other and then with the real data.

Keywords: lineplot, boxplot, distplot, violinplot, LSTM,ARIMA,GRU

1.GİRİŞ

Elektrik tüketimi son yıllarda yaşanan sıkıntılardan birisidir. Elektrik tüketimi hızla arttığından dağılımın doğru bir şekilde bazı aşamalardan geçip tahmin edilmesi gerekmektedir. Elektrik tüketiminde keskin yani doğru bir sonuç elde edebilmemiz için elektrik kullanımını takip etmemiz veya izlememiz gerekmektedir. Gözlemler sonucunda yapacağımız doğru bir tahmin bizi ileriki plansız elektrik tüketimlerini göz önüne alarak gereksiz elektrik dağıtımı önlenebilir. Ancak elektrik tüketimini etkileyen faktörlerin kullanılması sonucunda öngörülemeyen karmaşık bir tahmin modeli oluşabilir. Elektrik tüketimi zamana bağlı bir yapıdadır. Bu nedenle Elektrik tüketiminin tahmin modelini oluşturmak için belli başlı zaman serilerini kullanan yaklaşımlar vardır. Geçmişte kullanılan veriler, zaman serisi analizine dayalı çözümler ile zamana bağlı değişiklikleri ekrana yansıtır. Elektrik tüketimi tahminleri kısa vadeli(saatlik ile 1 hafta arası), orta vadeli (bir hafta ile bir yıl arası), uzun vadeli (bir yıldan fazla) olarak üç modelden oluşmaktadır.



Bu modelleri oluşturma aşamasında yardımımıza Derin Öğrenme yöntemleri ve Makine Öğrenimi Yöntemleri karşımıza çıkıyor. Derin Öğrenme ve Makine Öğrenimi yöntemlerinin çoğu zaman tahmine dayalı modelleme problemleri için bir çözüm anahtarı olduğu bilinmekte. Derin Öğrenme, bilgisayarlara insanların doğal olarak gelen bir şeyi yaptırmayı öğreten bir makine öğrenimi tekniğidir.

2. GENEL BİLGİLER

2.1 Veri Seti

Elektrik tüketimi konusunda tahmin modellerini yapabilmemiz için öncelikle veri setlerine ihtiyacımız bulunmaktadır. Kullanacağımız veri setleri 2011-2013 yılları arasında tüketilen elektrik tüketimi ile 2006-2011 yılları arasında enerji tüketimini gösteren veri setleri ele alınmıştır. Veri setlerimizin tek değişkenli veri seti tam “66912” veriden oluşmakta ve bu veri setimiz bir “csv” dosya uzantısı şeklindedir. Çok değişkenli veri setinde iki girişin toplam verisi sayısı “2884” olarak hesaplandı. Veri seti hazırlanması Derin Öğrenme yönteminin ilk başlarında gelir. Veri seti ne kadar düzenli ise yapılacak işlemler o kadar daha kesin veya daha doğru sonuç elde edilmeye olanak sağlar.

Tek değişkenli veri setinin ilk 5 satırının görseli aşağıdaki gibidir.

1	Office,,,
2	Date,Values,00:00,00:15,00:30,00:45,01:00,01:15,01:30,01:45,02:00,02:15,02
3	29/03/2011,96,nan,nan,nan,nan,nan,nan,nan,nan,nan,nan,nan,nan,nan,nan,na
4	30/03/2011,96,32.10000229,32.40000153,33.60000229,33,29.70000076,30.
5	31/03/2011,96,32.10000229,33.60000229,35.10000229,33.45000076,31.200

Veriler aşağıya ve yana doğru devam etmektedir.

Çok değişkenli veri setinin ilk 5 satırının görseli aşağıdaki gibidir.

1	datetime,Global_active_power,Global_reactive_power,Voltage,Global_intensity,Sub_metering_1,Sub_metering_2,Sub_metering_3,Sub_metering_4
2	2006-12-16,1209.1760000000004,34.92199999999976,93552.53000000006,5180.800000000008,0.0,546.0,4926.0,14680.933319299998
3	2006-12-17,3390.46,226.0059999999994,345725.32000000024,14398.59999999998,2033.0,4187.0,13341.0,36946.66673200004
4	2006-12-18,2203.825999999997,161.79199999999966,347373.64000000036,9247.199999999988,1063.0,2621.0,14018.0,19028.43328100003
5	2006-12-19,1666.1940000000006,150.94199999999978,348479.0100000004,7094.000000000014,839.0,7602.0,6197.0,13131.900043499994

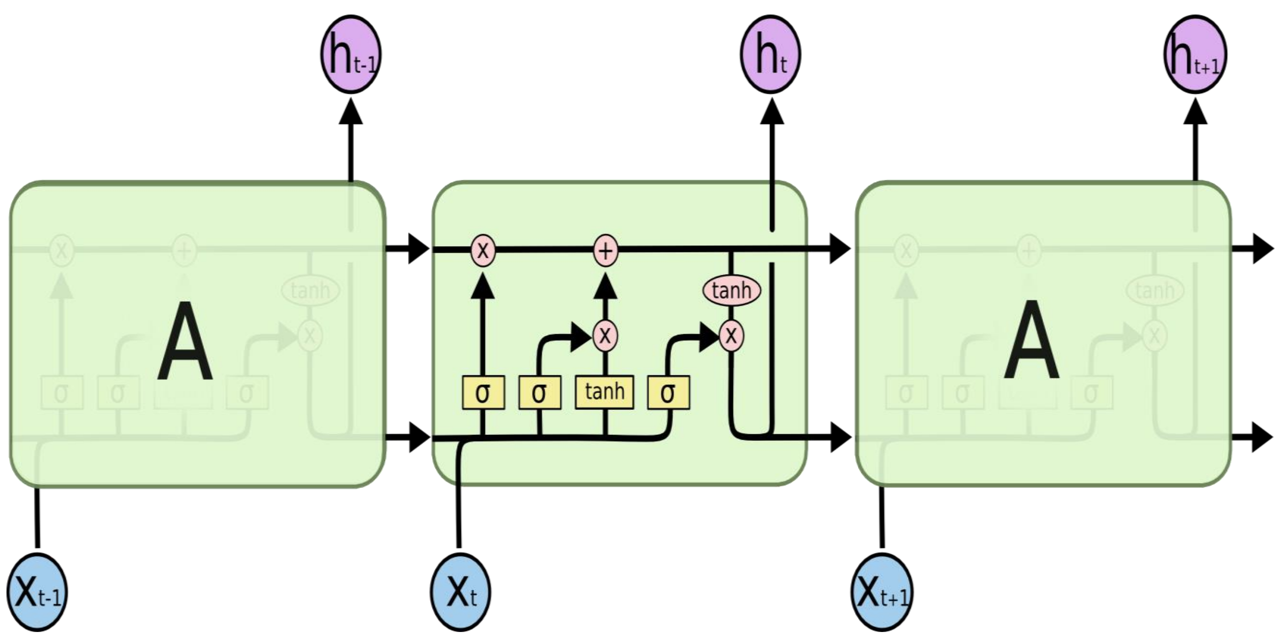
Veriler aşağıya doğru devam etmektedir.

3. MATERYAL VE YÖNTEM

3.1 LSTM MODELİ

Uzun Kısa Süreli Bellek ağları - genellikle "LSTM" olarak adlandırılır - uzun vadeli bağımlılıkları öğrenebilen özel bir RNN türüdür. LSTM'ler, uzun vadeli bağımlılık sorununu önlemek için açıkça tasarlanmıştır. Bilgiyi uzun süre hatırlamak, pratik olarak onların varsayılan davranışıdır, öğrenmek için mücadele ettikleri bir şey değildir.

Örnek bir LSTM modeli aşağıdaki gibi çalışır.



3.1.1 Kütüphaneler ve Veri seti İşlemleri

In [1]:
```
import csv
import math
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import random
import statsmodels.api as sm
import statsmodels.formula.api as smf
import tensorflow as tf
import seaborn as sns
from math import sqrt
from numpy.random import seed
seed(1)
from pandas import DataFrame
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from statsmodels.tsa.stattools import adfuller
from tensorflow.keras.layers import Dense
from tensorflow.keras import layers
from tensorflow.keras.layers import LSTM
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

- Öncelikle gerekli kütüphaneleri import ediyoruz ve as ile kısaltıyoruz. İmport ettiğimiz kütüphaneler sırasıyla; matplotlib, numpy, pandas, statsmodels, tensorflow, seaborn, sklearn şeklinde devam ediyor.

Verileri İçeri Aktarma Ve Ekrana Gösterme

In [2]:
```
# Tekrarlanabilirlik için rastgele tohum düzeltme metodunu kullanalım.
np.random.seed(7)

# veri kümesini yükleyelim
df = pd.read_csv('dccelectricitycivicsblocks34p20130221-1840.csv', engine='python', skipfooter=3)
df.head(10)
```

Out[2]:

	Office	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 88	Unna
0	Date	Values	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	...	21:30	
1	29/03/2011	96	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	31.80000305	31.2000
2	30/03/2011	96	32.10000229	32.40000153	33.60000229	33	29.70000076	30.90000153	31.50000191	33	...	35.70000076	33.6000
3	31/03/2011	96	32.10000229	33.60000229	35.10000229	33.45000076	31.20000076	31.35000038	34.20000076	33.75	...	33	
4	01/04/2011	96	32.70000076	34.5	30.30000305	33	33	31.5	30.30000305	32.84999847	...	32.25	33.4500
5	02/04/2011	96	31.80000114	32.40000153	30.45000076	33.90000153	33.30000305	30.75000191	30.75	29.85000038	...	34.5	31.8000
6	03/04/2011	96	32.85000229	30.30000114	32.10000229	30	34.05000305	30.75000191	32.10000229	28.80000114	...	34.95000076	31.8000
7	04/04/2011	96	32.25000381	35.40000153	37.34999847	34.35000229	34.95000076	36.45000076	36.15000153	33.15000153	...	32.55000305	31.6500
8	05/04/2011	96	30.90000153	32.85000229	32.10000229	35.55000305	32.55000305	33	31.80000114	31.5	...	33.30000305	
9	06/04/2011	96	32.55000305	30.30000114	33	29.70000076	33.60000229	32.70000076	28.95000076	28.20000076	...	33.75	

- Np.random.seed(7) ile rastgele sayı ürettik.
- Pd.read_csv ile csv dosyasının adını tanımladık ve df'ye eşitledik.
- Df.head(10) diyerek veri setinde ilk 10 veri satırını ekrana yazdırmasını sağladık.

In [3]:

```
df2=df.rename(columns=df.iloc[0])
df3=df2.drop(df.index[0])
df3
df3.drop(df3.index[0])
df4=df3.drop('Date', axis=1)
df5=df4.drop('Values', axis=1)
df5
df6=df5.dropna()
df7=df6.values
df7
dataset=np.sum(df7, axis=1, dtype=float)
dataset
```

- Öncelikle veri setinde ilk satırı rename ile sildik ve df2'ye eşitledikten sonra drop metodu ile çıkarttık ve df3'e eşitledik.
- Bu sefer df3'deki ilk satırı çıkarttık ve Date ve Values yazısı bir string olduğu için onlarıda ayrı ayrı çıkarttık ve df5'e eşitledik.
- Df5' üzerinde dropna metodu ile null değerleri çıkarttık ve df6'ya eşitledik.
- Oluşan df6'yı düzenli bir şekilde ayarlamak için values metodu kullanıldı ve df7ye eşitlendi.
- Oluşan df7'dede np.sum metodu ile aynı satır üzerindeki dizileri topladık ve dataset'e eşitledik. Aynı satırda toplamamız bize günlük enerjiyi verecektir.

Df3 'ün görüntüsü aşağıdaki gibidir.

In [4]:

```
df3
```

Out[4]:

	Date	Values	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	...	21:30	21:4
1	29/03/2011	96	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	31.80000305	31.2000007
2	30/03/2011	96	32.10000229	32.40000153	33.60000229	33	29.70000076	30.90000153	31.50000191	33	...	35.70000076	33.6000022
3	31/03/2011	96	32.10000229	33.60000229	35.10000229	33.45000076	31.20000076	31.35000038	34.20000076	33.75	...	33	3
4	01/04/2011	96	32.70000076	34.5	30.30000305	33	33	31.5	30.30000305	32.84999847	...	32.25	33.4500007
5	02/04/2011	96	31.80000114	32.40000153	30.45000076	33.90000153	33.30000305	30.75000191	30.75	29.85000038	...	34.5	31.8000011
...	...	...	...	...	...	...	...	...	...	...	...	...	.
688	13/02/2013	96	25	25	25.5	24.5	25	26	25.5	25.5	...	33	32.
689	14/02/2013	96	26	25.5	26	25.5	25	27	25.5	26.5	...	31	3
690	15/02/2013	96	25.5	24.5	25.5	24.5	25	26.5	25	25.5	...	29	2
691	16/02/2013	96	25	25	24.5	24.5	24.5	26.5	25	25.5	...	24.5	2
692	17/02/2013	96	24.5	24	24	24.5	24	24	24.5	24	...	25	2

692 rows × 98 columns

Df5 ‘in görüntüsü aşağıdaki gibidir.

```
In [5]: 1 df5
```

Out[5]:

	00:00	00:15	00:30	00:45	01:00	01:15	01:30	01:45	02:00	02:15	...	21:30	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	31.80000305	31.2
2	32.10000229	32.40000153	33.60000229	33	29.70000076	30.90000153	31.50000191	33	31.5	30.00000191	...	35.70000076	33.6
3	32.10000229	33.60000229	35.10000229	33.45000076	31.20000076	31.35000038	34.20000076	33.75	34.65000153	35.25	...	33	
4	32.70000076	34.5	30.30000305	33	33	31.5	30.30000305	32.84999847	33.15000153	32.70000076	...	32.25	33.4
5	31.80000114	32.40000153	30.45000076	33.90000153	33.30000305	30.75000191	30.75	29.85000038	28.80000114	33.45000076	...	34.5	31.8
...	...	...	...	...	...	...	...	...	...	...	...	...	
688	25	25	25.5	24.5	25	26	25.5	25.5	25	25.5	...	33	
689	26	25.5	26	25.5	25	27	25.5	26.5	26	27	...	31	
690	25.5	24.5	25.5	24.5	25	26.5	25	25.5	26	26	...	29	
691	25	25	24.5	24.5	24.5	26.5	25	25.5	25	25.5	...	24.5	
692	24.5	24	24	24.5	24	24	24.5	24	24	25.5	...	25	

Df7 ‘nin görüntüsü aşağıdaki gibidir.

```
In [6]: 1 df7
```

```
Out[6]: array([['32.10000229', '32.40000153', '33.60000229', ..., '32.70000076',
                '31.50000191', '33.60000229'],
               ['32.10000229', '33.60000229', '35.10000229', ..., '33.45000076',
                '32.10000229', '32.10000229'],
               ['31.80000114', '32.40000153', '30.45000076', ..., '31.5',
                '33.45000076', '32.55000305'],
               ...,
               ['25.5', '24.5', '25.5', ..., '25.5', '24.5', '25'],
               ['25', '25', '24.5', ..., '24.5', '25', '24'],
               ['24.5', '24', '24', ..., '25', '25', '25']], dtype=object)
```

```
In [7]: 1 tseries=dataset

In [8]: 1 tseries.shape
Out[8]: (680,)
```

- Sonraki adımda dataseti tseries’e eşitliyoruz ve shape metodu ile veri sayısını kontrol ediyoruz.

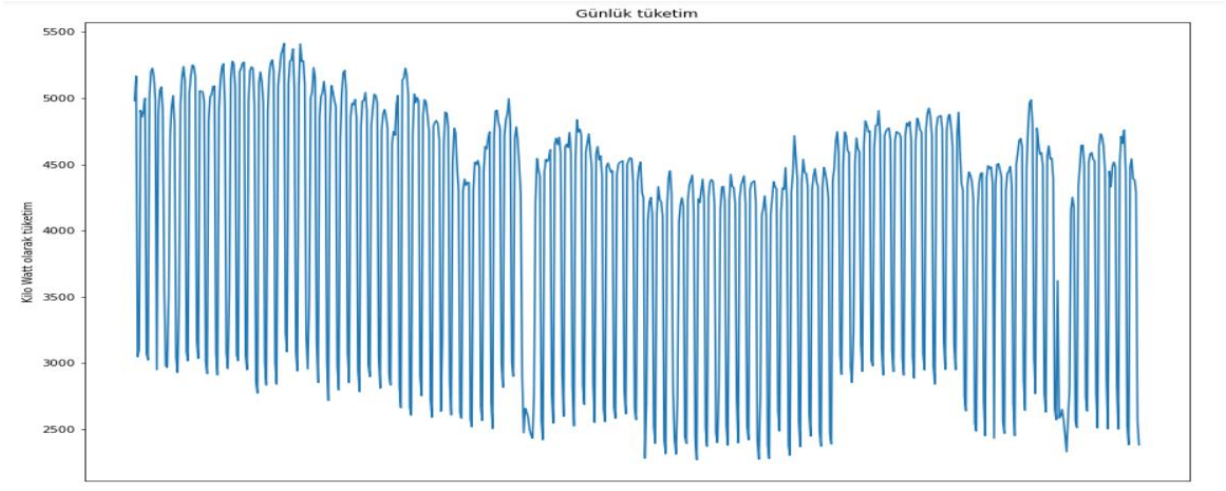
3.1.2 Grafikler

Grafikler, çeşitli yapay ve doğal süreçler tarafından üretilen zengin ve karmaşık verileri temsil etmek için kullanılan bir araçtır. Bir grafik , düğümlere (bilgiyi tutan varlıklar) ve kenarlara (aynı zamanda bilgiyi tutan düğümler arasındaki bağlantılar) sahip olan ve bu nedenle, ilişkisel bir doğanın yanı sıra bir bileşim niteliğine sahip olan yapılandırılmış bir veri türü olarak düşünülebilir. Grafik, verileri yapılandırmanın bir yoludur, ancak kendi başına bir veri noktası da olabilir. Grafikler, bir Öklid dışı veri türüdür , yani görüntüler, metin ve ses gibi diğer veri türlerinin aksine 3B olarak bulunurlar. Grafikler, üzerlerinde gerçekleştirilebilecek olası eylemleri ve analizleri sınırlayan belirli özelliklere sahip olabilir. Bu bölümde veri setimizi grafiklere dökmeyi göstereceğiz.

In [9]:
```
plt.figure(figsize=(15,10))
plt.plot(tseries)
plt.tick_params(
    axis='x',          # değişiklikler x ekseni için geçerlidir
    which='both',      # hem büyük hem de küçük keneler etkilenir.
    bottom=False,      # alt kenar boyunca işaretler kapalı
    top=False,         # üst kenardaki keneler kapalı
    labelbottom=False) # alt kenardaki etiketler kapalı
plt.ylabel('Kilo Watt olarak tüketim')
plt.title("Günlük tüketim")
plt.show()
```

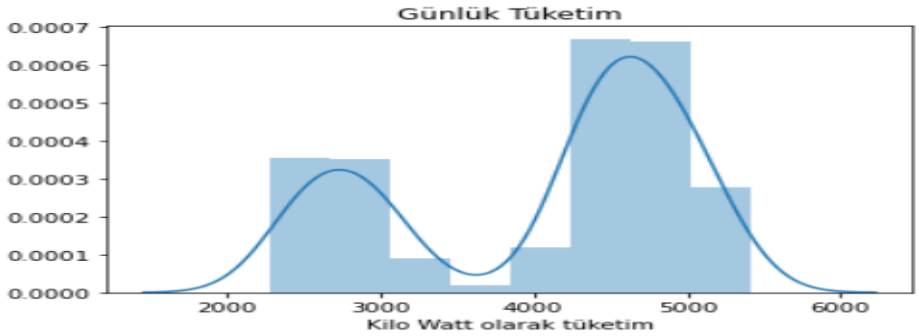
- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Tseries'deki verileri plt.plot'a tanımladık.
- Tick_params metodu ile işaretlerin, işaret etiketlerinin ve kılavuz çizgilerinin görünümünü değiştirmek için kullanıldık ve özelliklerini tanımladık.
- Y ekseni için bir etiket belirledik bu etiket 'Kilo Watt olarak tüketim' şeklindedir.
- Grafiğe title ile bir başlık oluşturduk bu başlık 'Günlük tüketim' şeklindedir.
- Son olarak plt.show ile grafiğin göstermesi sağladık.

Çıktısı aşağıdaki gibidir;



```
In [10]:
1  #Histogram Grafiği
2  sns.distplot(tseries)
3  plt.title("Günlük Tüketim")
4  plt.xlabel('Kilo Watt olarak tüketim')
```

```
Out[10]: Text(0.5, 0, 'Kilo Watt olarak tüketim')
```



- Histogram grafiği için öncelikle seaborn kütüphanesini kullandık.
- Sns.distplot modülü üzerinde çizgi bulunan bir histogram grafiği göstermemizi sağlar. Bu modül ile tseries veri setimizin değerlerini histogram grafiğine aktarmış olundu.
- Başlık olarak title ile ‘Günlük Tüketim’ yazısı yazıldı.
- X eksenine bir etiket olarak ‘Kilo Watt olarak tüketim yazıldı.’

3.1.3 Test ve Train İşlemleri

```
In [11]:
1  df=pd.DataFrame(tseries)
2  df
```

```
Out[11]:
```

	0
0	4981.500193
1	5166.600164
2	3046.350145
3	3101.100138
4	4908.600164
...	...
675	4390.000000
676	4385.000000
677	4289.500000
678	2564.000000
679	2383.000000

- Tseries veri setini DataFrame metodu ile tabloya aktardık ve df’ye eşitledik.
- ‘df’ yazarak direk ekrana gözükmesini sağladık.

```
In [12]:  1  df=np.array(df)
```

- df'yi Numpy Array'a dönüştürdük ve tekrardan df'ye eşitledik.

```
In [13]:  1  train_size = int(len(df) * 0.8)
          2  test_size = len(df) - train_size
          3  train, test = df[0:train_size,:], df[train_size:len(df),:]
```

- Bu bölümde train_size ve test_size bölümü oluşturduk ve veri setinde satır-sütun işlemi olarak kullanıp train ve test'e eşitledik.

```
In [15]:  1  df
Out[15]: array([[4981.5001927 ],
                [5166.60016445],
                [3046.35014537],
                [3101.10013769],
                [4908.60016439],
                [4858.50017742],
                [4905.00019836],
                [4999.95019526],
                [3075.90013122],
                [3023.5501442 ],
                [5004.6001587 ],
                [5199.30019957],
                [5226.75017163],
                [5162.55022428],
                [4991.55017468],
                [2950.20010378],
                [4883.85017776],
                [5055.15017129],
                [5084.10021592],
                [4914.00019451]
```

- Numpy Array 'a dönüştürdüğümüz veri setini ekrana çıkarttık.

MinMaxScaler ile veri kümesini normalleştirelim

Training Data

```
In [16]:
1 scaler = MinMaxScaler(feature_range=(0, 1))
2 train = scaler.fit_transform(train)
3 train
```

- MinMaxScaler ile veri kümesini normalleştirdik. Train bölümünü fit transform ile eğitim setinin parametrelerin uydurulması sağlandı.

- ‘train’ yazarak ekrana çıktı vermesini sağladık. Çıktısı;

```
Out[16]: array([[0.86253241],
                [0.921413  ],
                [0.24695818],
                [0.26437424],
                [0.8393428 ],
                [0.82340591],
                [0.83819764],
                [0.86840138],
                [0.25635808],
                [0.23970547],
                [0.86988054],
                [0.93181493],
                [0.94054681],
                [0.92012471],
                [0.86572932],
                [0.21637271],
                [0.83146979],
                [0.88596058],
                [0.89516964],
```

Test Data

```
In [17]:
1 test = scaler.fit_transform(test)
2 test
```

```
Out[17]: array([[0.95295446],
                [0.95558901],
                [0.91738803],
                [0.31802785],
                [0.23372224],
                [0.91305984],
                [0.95013173],
                [0.95859992],
                [0.92096349],
                [0.86789612],
                [0.30466692],
                [0.2331577 ],
                [0.89593527],
                [0.96499812],
                [0.85001882],
                [0.76176139],
                [0.74275499],
                [0.15619119],
                [0.11610839],
                [0.75442228]
```

- Aynı işlemi test içinde yaptık ve test yazarak ekrana çıktı vermesini sağladık.

3.1.4 LSTM Modeli Oluşturma

LSTM modelini yapılandıralım

```
In [18]:
1 #Yeniden inceleme süresi
2 lookback = 5
3 X_train, Y_train = create_dataset(train, lookback)
4 X_test, Y_test = create_dataset(test, lookback)
```

- Öncelikle lookback değerini 5 olarak belirliyoruz bu işlem daha performanslı çalışması için yapıldı.
- X_train ve Y_train oluşturuldu.
- X_test ve Y_test oluşturuldu.

```
In [19]:
1 X_train
```

```
Out[19]: array([[0.86253241, 0.921413  , 0.24695818, 0.26437424, 0.8393428 ],
                [0.921413  , 0.24695818, 0.26437424, 0.8393428 , 0.82340591],
                [0.24695818, 0.26437424, 0.8393428 , 0.82340591, 0.83819764],
                ...,
                [0.79366336, 0.8323127 , 0.84471866, 0.82467826, 0.77903069],
                [0.8323127 , 0.84471866, 0.82467826, 0.77903069, 0.2258521 ],
                [0.84471866, 0.82467826, 0.77903069, 0.2258521 , 0.18179503]])
```

- X_train yazarak ekrana verileri gösterildi.

```
In [20]:
1 Y_train
```

```
Out[20]: array([0.82340591, 0.83819764, 0.86840138, 0.25635808, 0.23970547,
                0.86988054, 0.93181493, 0.94054681, 0.92012471, 0.86572932,
                0.21637271, 0.83146979, 0.88596058, 0.89516964, 0.84106056,
                0.41720613, 0.22634519, 0.22171681, 0.39625915, 0.77969876,
                0.84673867, 0.87479522, 0.80460611, 0.24595616, 0.20964486,
                0.36691428, 0.85236906, 0.91773892, 0.94436404, 0.90948419,
                0.26050931, 0.23774913, 0.88176165, 0.92274903, 0.94822898,
                0.9445549 , 0.9204587 , 0.2861801 , 0.24323639, 0.88624689,
                0.88514943, 0.88376569, 0.85943092, 0.23641311, 0.20659109,
                0.81434003, 0.87093029, 0.87732413, 0.89660108, 0.89779396,
                0.25444947, 0.20353731, 0.84411434, 0.91191768, 0.94274172,
                0.95123503, 0.85303708, 0.25678752, 0.21890162, 0.40231898,
                0.91535317, 0.957104  , 0.95075787, 0.89197271, 0.25201599,
                0.23794   , 0.93157636, 0.9404991 , 0.95280964, 0.95514767,
                0.25025053, 0.21642042, 0.8609578 , 0.93424841, 0.94355289,
                0.93916307, 0.85208277, 0.18736184, 0.16011644, 0.90285178,
                0.9313855 , 0.90733703, 0.85575684, 0.23436135, 0.17958426,
                0.83853165, 0.92661397, 0.95266649, 0.96011008, 0.93219665,
                0.23397962, 0.18177916, 0.89860511, 0.93472555, 0.97385204,
                0.98420627, 1.        , 0.30450275, 0.25903013, 0.90332894,
```

- Y_train yazarak ekrana verileri gösterildi.

```
In [21]:
#Girdiyi [örnekler, zaman adımları, özellikler] olacak şekilde yeniden şekillendirelim
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

- X_train ve X_test girdimzi örnekler, zaman adımları, özellikler olacak şekilde yeniden şekillendirdik.

```
In [22]:
X_train.shape[0]
```

```
Out[22]: 538
```

```
In [23]:
X_train.shape[1]
```

```
Out[23]: 1
```

- Shape metodunu kullanarak shape[0] ile X_train'deki veri sayısını ekrana yazdırdık.
- Shape metodunu kullanarak shape[1] ile X_train'deki veri sayısını ekrana yazdırdık.

```
In [24]:
# LSTM ağı oluşturalım
model = tf.keras.Sequential()
model.add(LSTM(4, input_shape=(1, lookback)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
history=model.fit(X_train, Y_train, validation_split=0.2, epochs=100, batch_size=1, verbose=2)
```

```
Epoch 1/100
430/430 - 4s - loss: 0.1852 - val_loss: 0.0777
Epoch 2/100
430/430 - 0s - loss: 0.0897 - val_loss: 0.0724
Epoch 3/100
430/430 - 0s - loss: 0.0836 - val_loss: 0.0690
Epoch 4/100
430/430 - 0s - loss: 0.0772 - val_loss: 0.0595
Epoch 5/100
430/430 - 0s - loss: 0.0693 - val_loss: 0.0563
Epoch 6/100
430/430 - 0s - loss: 0.0629 - val_loss: 0.0484
Epoch 7/100
430/430 - 0s - loss: 0.0577 - val_loss: 0.0430
Epoch 8/100
430/430 - 0s - loss: 0.0534 - val_loss: 0.0398
Epoch 9/100
430/430 - 0s - loss: 0.0503 - val_loss: 0.0368
```

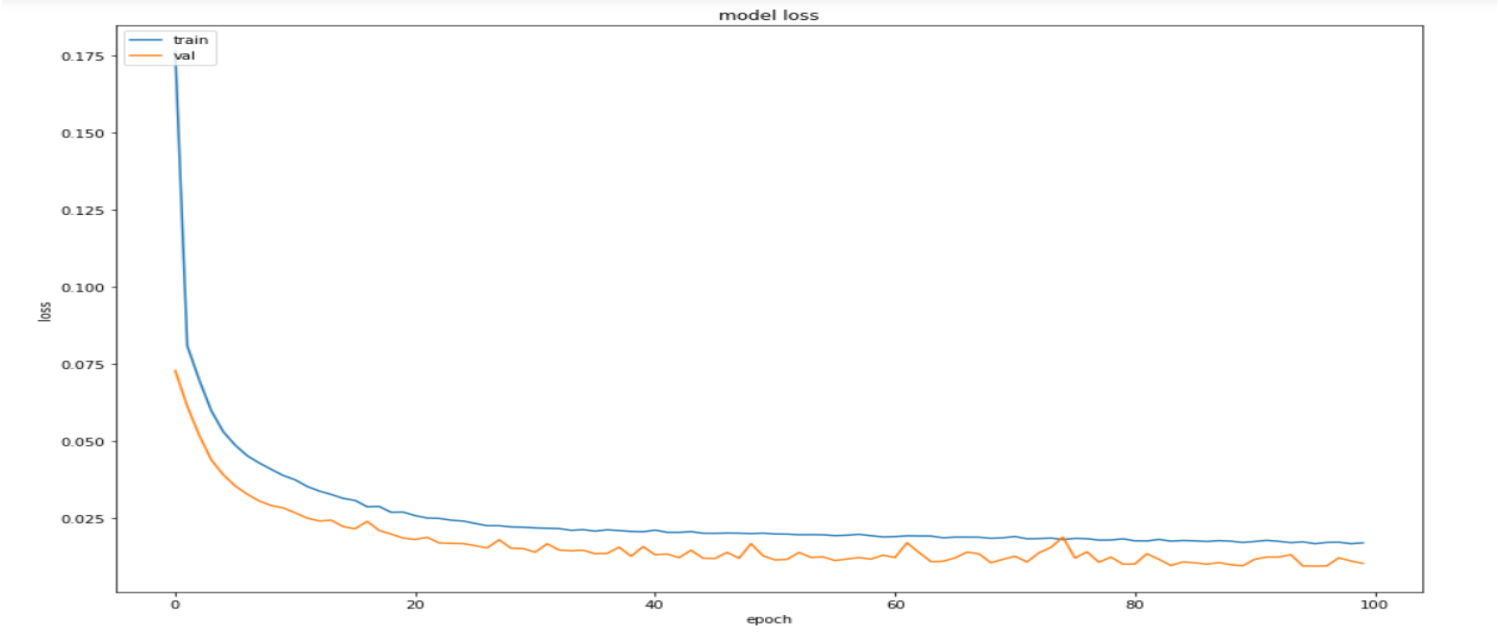
- Öncelikle keras kütüphanesinden Sequential'i model'e eşitledik.
- Model.add diyerek 4 katmanlı lstm modeli oluşturduk.
- Epoch, bir TÜM veri kümesinin yalnızca BİR KEZ sinir ağından ileri ve geri aktarılmasıdır. Biz 100 kez ileri geri aktardık.

- batch_size, öğrenme sürecinde ağı eğitmek için kullanılacak olan eğitim örneğinizin alt küme boyutunu belirtir.
- Bu işlemi history'e eşitledik çünkü val-loss grafiği için önemli.

In [25]:
```
plt.figure(figsize=(15,10))
#history'deki tüm verileri listeleyelim
print(history.history.keys())
# Doğruluk için Geçmiş verileri düzenleme yapalım
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Print içerisinde histor.history.keys() diyerek historydeki tüm verileri listeledik.
- Plt.plot ile loss ve val_loss tanımlaması yaptık.
- Daha sonra model loss başlığı attık.
- Y ekseni için loss, X ekseni için epoch etiketi yazdırdık.
- Son olarak plt.legend ile çalıştırma işlemi yapıp show ile ekrana gösterdik.

Çıktısı;



```
In [26]:
1 # Oluşturduğumuz Tahminleri Eşitleyelim
2 trainpred = model.predict(X_train)
3 testpred = model.predict(X_test)
```

- Bu adımda Oluşturduğumuz Tahminleri Eşitledik.
- X_train'i trainpred'e, X_test'i testpred'e eşitledik.
-

```
In [27]:
1 trainpred

Out[27]: array([[ 0.82194835],
                [ 0.6889534 ],
                [ 0.7687059 ],
                [ 0.6419595 ],
                [ 0.0803197 ],
                [ 0.66745996],
                [ 0.8615855 ],
                [ 0.9529891 ],
                [ 0.9594952 ],
                [ 0.6837667 ],
                [ 0.19774047],
                [ 0.2741551 ],
                [ 0.7065216 ],
                [ 0.678859  ],
                [ 0.4843637 ],
                [ 0.57477367],
                [ 0.11456878],
                [ 0.6156937 ],
                [ 0.6769189 ],
                [ 0.46207199]
```

- 'trainpred' yazarak ekrana verileri yansıttık.

```
In [28]:
1 testpred

Out[28]: array([[ 0.7937875 ],
                [ 0.8794083 ],
                [ 0.9538952 ],
                [ 0.93571615],
                [ 0.65536195],
                [ 0.19400027],
                [ 0.2729481 ],
                [ 0.7833959 ],
                [ 0.8675741 ],
                [ 1.0162852 ],
                [ 0.73148054],
                [ 0.30693814],
                [ 0.27202633],
                [ 0.3533358 ],
                [ 0.7179526 ],
                [ 0.82344484],
                [ 0.80959344],
                [ 0.77751404],
                [ 0.5950098 ],
                [ 0.21036318]
```

- 'testpred' yazarak ekrana verileri yansıttık.

```
In [29]:
# Tahminleri normal değerlere dönüştürelim
trainpred = scaler.inverse_transform(trainpred)
Y_train = scaler.inverse_transform([Y_train])
testpred = scaler.inverse_transform(testpred)
Y_test = scaler.inverse_transform([Y_test])
LSTMtahmin = testpred
```

- Bu bölümde dönüştürme işlemi uyguladık.
- trainpred, Y_train, testpred, Y_test'i dönüştürdük ve testpred'i LSTMtahmine eşitledik.

```
In [30]: Y_train.shape
Out[30]: (1, 538)

In [31]: Y_test.shape
Out[31]: (1, 130)

In [32]: X_train.shape
Out[32]: (538, 1, 5)

In [33]: X_test.shape
Out[33]: (130, 1, 5)
```

- Bu bölümde sırasıyla Y_train, Y_test, X_train, X_testlerin veri sayısını shape ile ekrana yazdırdık.

```
In [34]: Y_train
Out[34]: array([[4518.28951509, 4557.59113815, 4637.8424781 , 3011.6434176 ,
                 2967.39742769, 4641.77260773, 4806.33226731, 4829.53286879,
                 4775.27134202, 4630.74281586, 2905.40228055, 4539.71522927,
                 4684.49726678, 4708.96572332, 4565.19789723, 3439.01669686,
                 2931.89915677, 2919.60157603, 3383.3605489 , 4402.1595992 ,
                 4580.28464288, 4654.83090304, 4468.33843892, 2984.00551222,
                 2887.52640548, 3305.39123188, 4595.24459172, 4768.93231222,
                 4839.67525246, 4746.9994882 , 3022.67323035, 2962.19943991,
                 4673.34069105, 4782.24416242, 4849.94440544, 4840.18237672,
                 4776.15877125, 3090.88053001, 2976.77908865, 4685.25797653,
                 4682.34204071, 4678.66542886, 4614.00795392, 2958.64964103,
                 2879.41252047, 4494.20145533, 4644.56177266, 4661.55021527,
                 4712.76906743, 4715.93854042, 3006.57223143, 2871.29863554,
                 4573.31180319, 4753.46528423, 4835.36475348, 4857.93146626,
                 4597.01951125, 3012.78443227, 2912.12160079, 3399.46152373,
                 4762.59336455, 4873.52533611, 4856.66365482, 4700.47148026,
                 3000.1064951 , 2962.70658022, 4805.69837847, 4829.40611879,
                 4862.11521194, 4868.32736766, 2995.4156686 , 2905.52906765,
                 4618.06488511, 4812.79801492, 4837.52002313, 4825.85627158,
                 4594.48392707, 2838.32041844, 2755.9293924 , 4729.3771777
```

- Y_train yazarak verileri görüntüledik.

```
In [35]: 1 Y_test

Out[35]: array([[4756.5, 4855. , 4877.5, 4777.5, 4636.5, 3140. , 2950. , 4711. ,
                 4894.5, 4589. , 4354.5, 4304. , 2745.5, 2639. , 4335. , 4442.5,
                 4417.5, 4370.5, 4240.5, 2549.5, 2486.5, 4184.5, 4369. , 4432. ,
                 4437. , 2722.5, 2451.5, 4403. , 4491. , 4468. , 4481.5, 4361.5,
                 2432. , 4340.5, 4498.5, 4506. , 4459.5, 4407. , 2562.5, 2469. ,
                 4304.5, 4430.5, 4449.5, 4484. , 4335. , 2695. , 2451. , 4500. ,
                 4581. , 4679. , 4696. , 4632.5, 2873. , 2643. , 4475. , 4771. ,
                 4965. , 4987.5, 4809. , 3042.5, 2769. , 4774.5, 4660.5, 4576. ,
                 4589. , 4491.5, 2774.5, 2630.5, 4564. , 4640. , 4540.5, 4547.5,
                 4399. , 2658.5, 2572.5, 3622.5, 2584.5, 2599.5, 2650. , 2576. ,
                 2460. , 2330.5, 2560.5, 2803. , 4152. , 4252. , 4177.5, 2566.5,
                 2512. , 4333.5, 4477.5, 4644.5, 4646.5, 4487.5, 2741.5, 2637. ,
                 4518. , 4579.5, 4589. , 4537. , 4526. , 2771. , 2510. , 4639. ,
                 4731. , 4709.5, 4638.5, 4454. , 2869.5, 2503.5, 4448.5, 4329.5,
                 4482. , 4518. , 4483. , 2840.5, 2502. , 4515. , 4711.5, 4658.5,
                 4759.5, 4450. , 2524.5, 2383.5, 4481.5, 4543. , 4390. , 4385. ,
                 4289.5, 2564. ]])
```

- Y_test yazarak verileri görüntüledik.

```
In [36]: 1 X_train

Out[36]: array([[[0.86253241, 0.921413  , 0.24695818, 0.26437424, 0.8393428 ]],

                [[0.921413  , 0.24695818, 0.26437424, 0.8393428 , 0.82340591]],

                [[0.24695818, 0.26437424, 0.8393428 , 0.82340591, 0.83819764]],

                ...,

                [[0.79366336, 0.8323127 , 0.84471866, 0.82467826, 0.77903069]],

                [[0.8323127 , 0.84471866, 0.82467826, 0.77903069, 0.2258521 ]],

                [[0.84471866, 0.82467826, 0.77903069, 0.2258521 , 0.18179503]]])
```

- X_train yazarak verileri görüntüledik.

```
In [37]: 1 X_test

Out[37]: array([[[0.95295446, 0.95558901, 0.91738803, 0.31802785, 0.23372224]],

                [[0.95558901, 0.91738803, 0.31802785, 0.23372224, 0.91305984]],

                [[0.91738803, 0.31802785, 0.23372224, 0.91305984, 0.95013173]],

                [[0.31802785, 0.23372224, 0.91305984, 0.95013173, 0.95859992]],

                [[0.23372224, 0.91305984, 0.95013173, 0.95859992, 0.92096349]],

                [[0.91305984, 0.95013173, 0.95859992, 0.92096349, 0.86789612]],

                [[0.95013173, 0.95859992, 0.92096349, 0.86789612, 0.30466692]],

                [[0.95859992, 0.92096349, 0.86789612, 0.30466692, 0.2331577 ]],

                [[0.92096349, 0.86789612, 0.30466692, 0.2331577 , 0.89593527]],

                [[0.86789612, 0.30466692, 0.2331577 , 0.89593527, 0.96499812]],
```

- X_test yazarak verileri görüntüledik.

Test Verilerinde Tahmin Doğruluğu

In [38]:
```
#RMSE'yi Hesaplayalım
trainScore = math.sqrt(mean_squared_error(Y_train[0], trainpred[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
valScore = math.sqrt(mean_squared_error(Y_test[0], testpred[:,0]))
print('Test Score: %.2f RMSE' % (valScore))
```

```
Train Score: 324.96 RMSE
Test Score: 414.18 RMSE
```

- Test verilerinde tahmin doğruluğu için RMSE'yi hesapladık.
- Math kütüphanesini bu işlem için import etmiştik.
- Y_train ve trainpred'i kullanarak trainScore'u hesapladık.
- Print ile ekrana 'Train Score:' diyerek sonucu ekrana yazdırdık.
- Y_test ve testpred'i kullanarak Test Score'u hesapladık ve valScore'a eşitledik.
- Print ile ekrana 'Test Score:' diyerek sonucu yazdırdık.

In [39]:
```
# Train Tahminleri
trainpredPlot = np.empty_like(df)
trainpredPlot[:, :] = np.nan
trainpredPlot[lookback:len(trainpred)+lookback, :] = trainpred
```

In [40]:
```
# Test Tahminleri
testpredPlot = np.empty_like(df)
testpredPlot[:, :] = np.nan
testpredPlot[len(trainpred)+(lookback*2)+1:len(df)-1, :] = testpred
```

- İlk olarak Train Tahminleri bölümünü oluşturuyoruz.
- Np.empty_like ile ilk başlarda belirttiğimiz df veri setine benzer yeni bir dizi oluşturduk ve bunu 'trainpredPlot'a eşitledik.
- Np.nan ile içerisini boş yaptık.
- Son olarak loocback ve len ile trainpred'in verilerini ekliyoruz ve sonucu trainpred'e eşitliyoruz.
- İkinci olarak Test Tahminleri bölümünü oluşturuyoruz.
- Np.empty_like ile ilk başlarda belirttiğimiz df veri setine benzer yeni bir dizi oluşturduk ve bunu 'testpredPlot'a eşitledik.
- Np.nan ile içerisini boş yaptık.
- Son olarak loocback ve len ile trainpred'in verilerini ekliyoruz ve sonucu testpred'e eşitliyoruz.

```
In [41]:
1 Y_test=Y_test.reshape(-1)
2 Y_test.shape
3 Y_test=pd.Series(Y_test)
```

- Y_test'in son düzenlemelerini yapmak için shape metodunu kullandık
- Pd.Series ile Y_test verisindeki verileri elde tuttuk ve bunu Y_test'e eşitledik.

```
1 Y_test
0      4756.5
1      4855.0
2      4877.5
3      4777.5
4      4636.5
       ...
125    4543.0
126    4390.0
127    4385.0
128    4289.5
129    2564.0
Length: 130, dtype: float64
```

- Y_test diyerek verileri ekrana yazdırdık.

```
In [43]:
1 LSTMtahmin=LSTMtahmin.reshape(-1)
2 LSTMtahmin.shape
3 LSTMtahmin=pd.Series(LSTMtahmin)

In [44]:
1 LSTMtahmin

Out[44]: 0      4439.593262
         1      4667.087891
         2      4864.999512
         3      4816.697754
         4      4071.796875
                   ...
         125    4890.939453
         126    4721.482422
         127    4405.038574
         128    4031.699219
         129    2950.128418
         Length: 130, dtype: float32
```

- LSTM tahmin verilerinin son düzenlemelerini yapmak için shape metodunu kullandık.
- Pd.Series ile LSTMtahmin verisindeki verileri elde tuttuk ve bunu Y_test'e eşitledik.
- LSTMtahmin yazarak verileri ekrana yazdırdık.

In [45]:
```
import numpy as np

def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Ortalama Yön Doğruluğu """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))
```

In [46]:
```
mda(Y_test, LSTMtahmin)
```

Out[46]: 0.9846153846153847

- Öncelikle bir sorun olmaması için numpy kütüphanesini import ettik.
- Ortalama yön doğruluğunu hesaplama için mda'yı kullandık ve hesapladık.
- Mda(Y_test,LSTMtahmin) diyerek hesaplama sonucunu yani doğruluk sonucunu ekrana yazdırdık.

In [47]:
```
mse = mean_squared_error(Y_test, LSTMtahmin)
rmse = sqrt(mse)
print('RMSE: %f' % rmse)
```

```
RMSE: 414.175778
```

In [48]:
```
np.mean(Y_test)
```

Out[48]: 3909.15

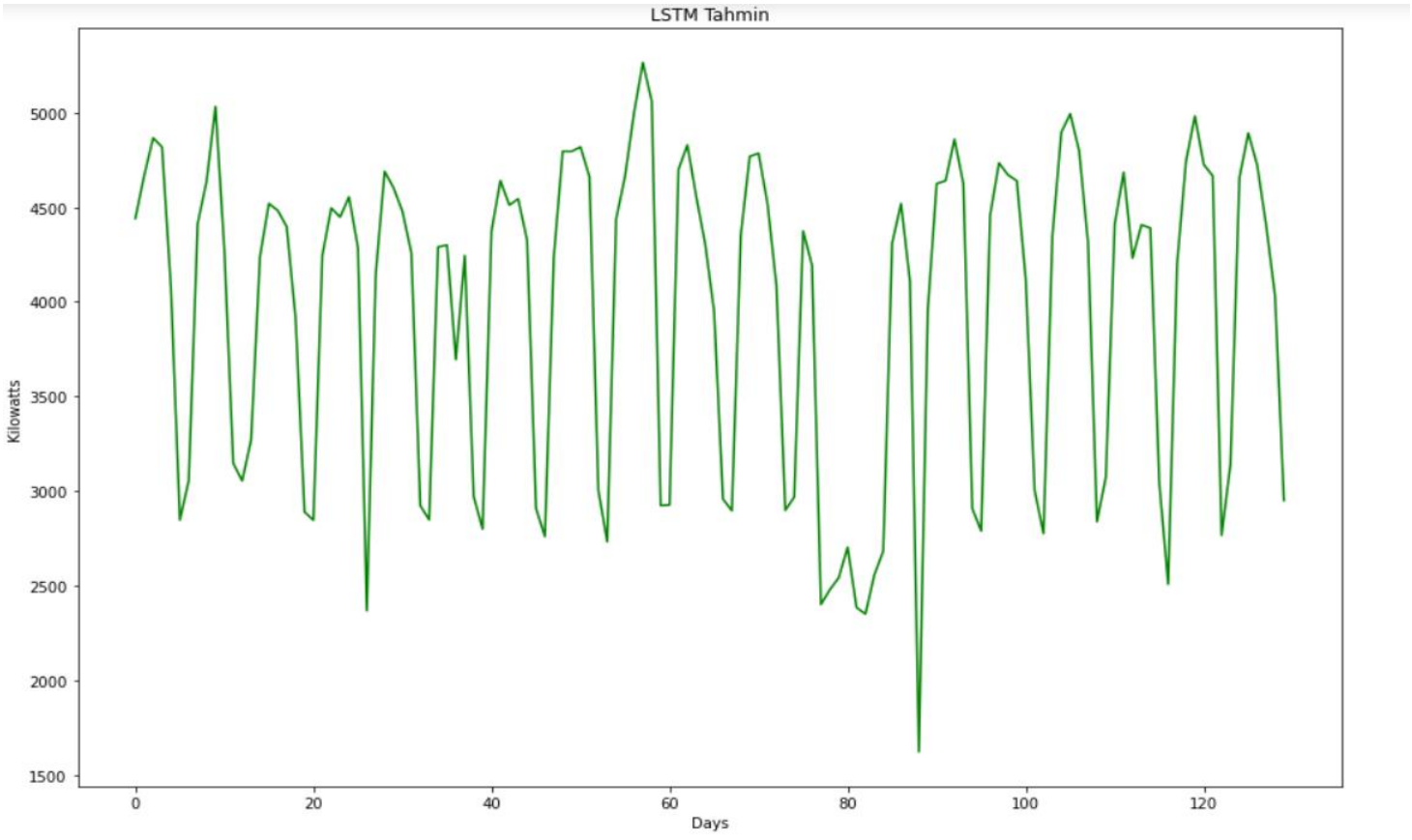
- Bu bölümde Y_test ve LSTMtahmin veri setlerinin mse'sini hesapladık.
- Daha sonra mse değerin sqrt ile karekökünü alınca rms değerini bulmuş olduk.
- Print ile 'RMSE:' şeklinde ekrana yazdırılıp hemen ardında sonucu ekrana yazdırdık.
- Diğer bölümde ise np.mean metodu ile Y_test'in verilerinin aritmetik ortalaması hesaplandı.

3.1.5 LSTM Modeli Karşılaştırılması

In [52]:
```
plt.figure(figsize=(15,10))
plt.plot(LSTMtahmin, color="green")
plt.xlabel('Days')
plt.ylabel('Kilowatts')
plt.title("LSTM Tahmin")
plt.show()
```

- Öncelikle LSTM tahmin modelinin grafiğini ekrana gösterelim.
- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plt.plot içerisine oluşturduğumuz LSTMtahmin veri setini yazıp renk olarak da yeşil rengi yazıyoruz.
- X ekseni üzerine etiket olarak ‘Days’ yani günler yazı etiketi atıyoruz.
- Y ekseni üzerin de etiket olarak ‘Kilowatts’ yazı etiketi atıyoruz.
- Plt.title ile başlık atarak başlık adımız LSTM Tahmin şeklinde yazıyoruz.
- Plt.show ile ekrana gösteriyoruz.

Çıktısı;

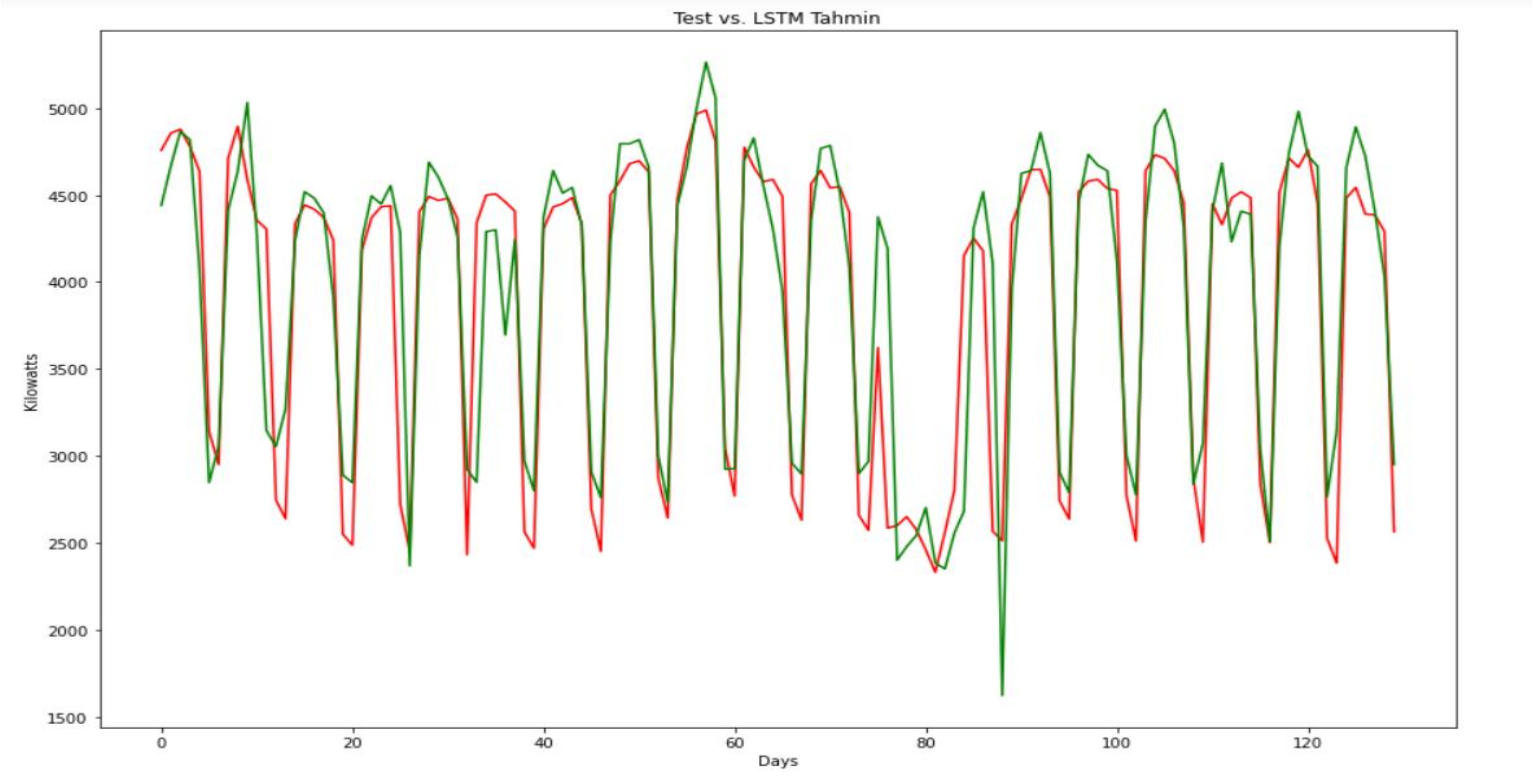


In [53]:

```
plt.figure(figsize=(15,10))
plt.plot(Y_test,color="red")
plt.plot(LSTMtahmin,color="green")
plt.xlabel('Days')
plt.ylabel('Kilowatts')
plt.title("Test vs. LSTM Tahmin")
plt.show()
```

- Tahmin modelimiz ile gerçek değerleri kıyaslamak için karşılaştırma tablosu yapacağız.
- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plt.plot içerisine oluşturduğumuz Y_test veri setini yazıp renk olarak da kırmızı rengi yazıyoruz.
- Plt.plot içerisine oluşturduğumuz LSTMtahmin veri setini yazıp renk olarak da yeşil rengi yazıyoruz.
- X ekseni üzerine etiket olarak 'Days' yani günler yazı etiketi atıyoruz.
- Y ekseni üzerin de etiket olarak 'Kilowatts' yazı etiketi atıyoruz.
- Plt.title kullanarak grafiğe başlık atıyoruz bu başlık 'Test vs LSTM Tahmin' şeklinde olacak.
- Daha sonra plt.show() ile ekrana karşılaştırma grafiğini yazıyoruz.

Çıktısı;



3.2 ARIMA Model

Zaman serileri yardımıyla tahmin yapmak için değişik yöntemler kullanılarak oluşturulan farklı modeller bulunmaktadır. Bu modeller arasında en çok bilinen ve yaygın olarak kullanılan ARIMA modelleridir. Seriyi oluşturan veriler arasında doğrusal bir ilişkinin olduğunu varsayan ve bu doğrusal ilişkiyi modelleyebilen ARIMA modelleri durağan ya da çeşitli istatistiksel yöntemlerle durağan hale getirilen zaman serilerine başarıyla uygulanabilmektedir. Oysa uygulamada karşılaşılan birçok zaman serisi sadece doğrusal ilişki içermemektedir. Yapısı gereği hem doğrusal hem de doğrusal olmayan ilişkileri modelleyebilen yapay sinir ağları (YSA) son yıllarda zaman serilerinin analizinde kullanılan alternatif yöntemlerden biri olmuştur.

Zaman serisi tahmini için popüler ve yaygın olarak kullanılan bir istatistiksel yöntemdir ARIMA model. Üstel yumuşatma ve ARIMA modelleri, zaman serisi tahmininde en yaygın kullanılan iki yaklaşımdır ve probleme tamamlayıcı yaklaşımlar sağlar. Üstel düzeltme modelleri, verilerdeki eğilim ve mevsimselliğin bir açıklamasına dayanırken, ARIMA modelleri verilerdeki otokorelasyonları tanımlamayı amaçlamaktadır.

Örnek bir LSTM modeli denklem mantığı aşağıdaki gibi çalışır.

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + .. + \phi_q \epsilon_{t-q}$$

Bir ARIMA modeli 3 terimle karakterize edilir: p, d, q

- p, AR teriminin sırasıdır.
- q, MA teriminin sırasıdır.
- d, zaman serisini durağan hale getirmek için gereken farklılaştırma sayısıdır.

d'nin değeri, seriyi durağan hale getirmek için gereken minimum farklılık sayısıdır. Ve eğer zaman serisi zaten sabitse, o zaman d = 0'dır.

Sonra, 'p' ve 'q' terimleri nelerdir?

'p', 'Otomatik Gerileyen' (AR) terimidir. Yordayıcı olarak kullanılacak Y gecikme sayısını ifade eder. Ve 'q', 'Hareketli Ortalama' (MA) teriminin sırasıdır. ARIMA Modeline girmesi gereken gecikmeli tahmin hatalarının sayısını ifade eder.

3.2.1 Veri Seti Grafikleri

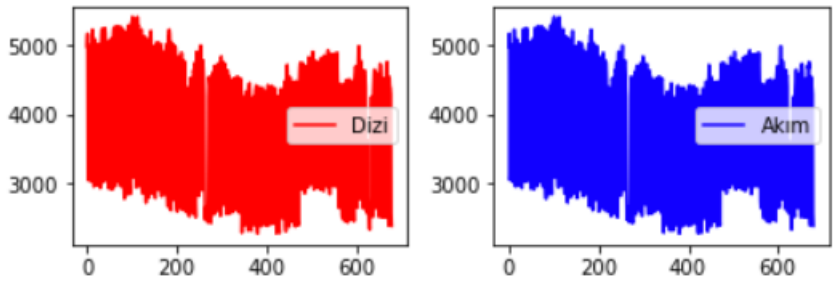
Grafikler, çeşitli yapay ve doğal süreçler tarafından üretilen zengin ve karmaşık verileri temsil etmek için kullanılan bir araçtır. Bir grafik , düğümlere (bilgiyi tutan varlıklar) ve kenarlara (aynı zamanda bilgiyi tutan düğümler arasındaki bağlantılar) sahip olan ve bu nedenle, ilişkisel bir doğanın yanı sıra bir bileşim niteliğine sahip olan yapılandırılmış bir veri türü olarak düşünülebilir. Grafik, verileri yapılandırmanın bir yoludur, ancak kendi başına bir veri noktası da olabilir. Grafikler, bir Öklid dışı veri türüdür , yani görüntüler, metin ve ses gibi diğer veri türlerinin aksine 3B olarak bulunurlar. Grafikler, üzerlerinde gerçekleştirilebilecek olası eylemleri ve analizleri sınırlayan belirli özelliklere sahip olabilir. Bu bölümde veri setimizi grafiklere dökmeyi göstereceğiz.

In [78]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition=seasonal_decompose(dataset, model='additive', freq=1)
trend=decomposition.trend
plt.subplot(221)
plt.plot(dataset,color='#ff0000', label='Dizi')
plt.legend(loc='best')
plt.subplot(222)
plt.plot(trend,color='#1100ff', label='Akım')
plt.legend(loc='best')
plt.tight_layout()

plt.show()
```

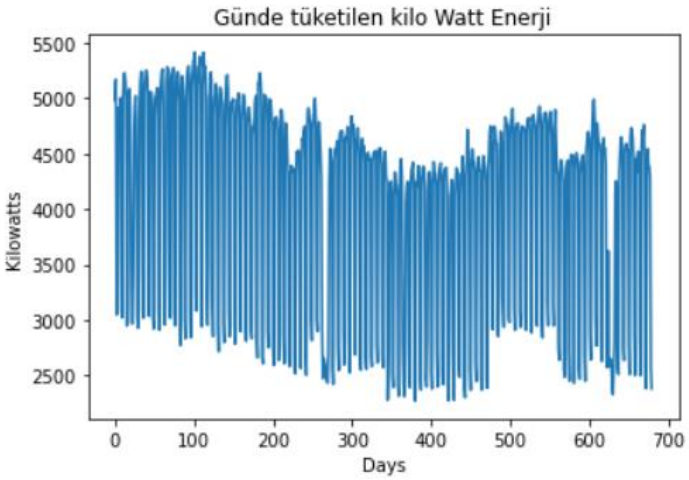
- İlk olarak statsmodels.tsa.seasonal kütüphanesini ve seasoal_decompose kütüphanesini import ettik.
- Statsmodels kütüphanesi, season_decompose () adı verilen bir işlevde naif veya klasik ayrıştırma yönteminin uygulamasını sağlar. Modelin eklemeli mi yoksa çarpan mı olduğunu belirlemenizi gerektirir.
- Seasonal_decompose içerisinde veri setinin hangi grafik şeklinde olması gerektiğini model içerisine yazdık. Yazdığımız ‘additive’ bir Katkı maddesi modelidir.
- Decomposition.trend ile ayrışma eğilimini trend’e eşitledik.
- Subplot ile ekseni belirliyoruz.
- Pyplot kütüphanesinden grafik oluşturmak için yardım istiyoruz ve plt.plot’un içerisine dataset(veri setimizi)’i giriyoruz ve daha sonra renk kodunu yazıp, Dizi adında etiket ekliyoruz.
- Plt.legend ile direk çalıştırıyoruz.
- Subplot ile ekseni belirliyoruz.
- Pyplot kütüphanesinden grafik oluşturmak için yardım istiyoruz ve plt.plot’un içerisine trend(ayrışma eğilimimizi)’i giriyoruz ve daha sonra renk kodunu yazıp, Akım adında etiket ekliyoruz.
- Plt.legend ile direk çalıştırıyoruz.

- Tighy_layout metodu ile grafik parametrelerini otomatik olarak ayarlıyoruz böylece grafikler şekil alanına sığacaktır.
- Plt.show diyerek grafikleri gösteriyoruz.

  Çıktısı;




```
In [79]:
1  plt.plot(dataset)
2  plt.xlabel('Days')
3  plt.ylabel('Kilowatts')
4  plt.title("Günde tüketilen kilo Watt Enerji")
5  plt.show()
```



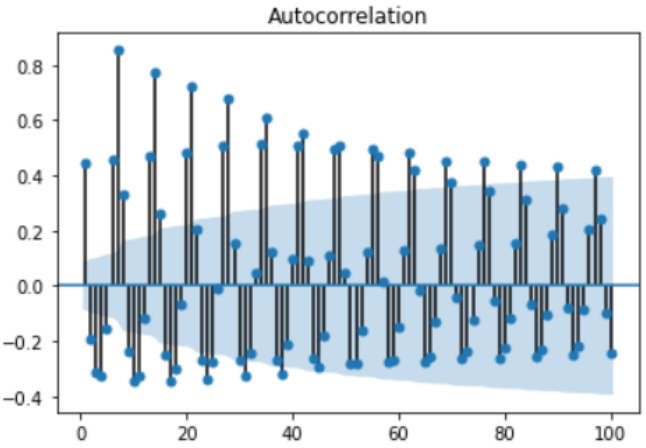
- Öncelikle Pyplot kütüphanesinden grafik oluşturmak için yardım istiyoruz ve plt.plot'un içerisine dataset(veri setimizi)'i giriyoruz.
- X ekseni üzerine label ile etiket atıyoruz ve etiketin adı 'Days' şeklinde olacak.
- Y ekseni üzerine label ile etiket atıyoruz ve etiketin adı 'Kilowatts' şeklinde olacak.
- Title ile grafiğe bir başlık ekliyoruz ve bu başlığımız 'Günde tüketilen kilo Watt Enerji' şeklinde.
- Plt.show diyerek ekrana gösteriyoruz grafiği.
- Yukardaki bölümde kodlarıyla beraber çıktısı gösterilmiştir.

```
In [80]:  1 train_df=dataset[:554]
          2 train_df
```

```
Out[80]: array([4981.5001927 , 5166.60016445, 3046.35014537, 3101.10013769,
                4908.60016439, 4858.50017742, 4905.00019836, 4999.95019526,
                3075.90013122, 3023.5501442 , 5004.6001587 , 5199.30019957,
                5226.75017163, 5162.55022428, 4991.55017468, 2950.20010378,
                4883.85017776, 5055.15017129, 5084.10021592, 4914.00019451,
                3581.55014991, 2981.55008892, 2967.00011064, 3515.70014566,
                4721.10016438, 4931.85019494, 5020.05018234, 4799.40017322,
                3043.20012856, 2929.05012318, 3423.45014192, 4949.55017475,
                5155.05015188, 5238.75021174, 5129.10016059, 3088.95013995,
                3017.40010454, 5041.95018196, 5170.80017096, 5250.90023994,
                5239.35021975, 5163.60019308, 3169.65013694, 3034.65012932,
                5056.05021094, 5052.6001988 , 5048.2501869 , 4971.75019264,
                3013.20013239, 2919.45011703, 4830.0002022 , 5007.90018087,
                5028.00018885, 5088.60017207, 5092.35015869, 3069.90011787,
                2909.85011099, 4923.60022544, 5136.75021744, 5233.65021135,
                5260.35019306, 4951.65018459, 3077.25013161, 2958.15011977,
                3534.75013923, 5147.55017665, 5278.80020143, 5258.85017209,
                5074.05018618, 3062.25013165, 3018.00013154, 5198.55020907,
                5226.60020638, 5265.30022429, 5272.65018464, 3056.7001438 ,
                2950.35011292, 4976.55018233, 5206.95019914, 5236.2002354
```

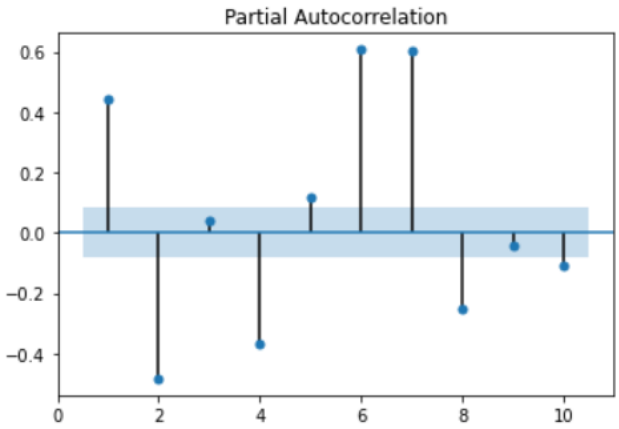
- Veri setinde ilk 554 sütunu seçiyoruz ve train_df'ye eşitliyoruz.
- Train_df yazarak ekrana veri kümesini gösteriyoruz.

```
In [81]:  1 from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
          2 plot_acf(train_df, lags=100, zero=False);
          3
```



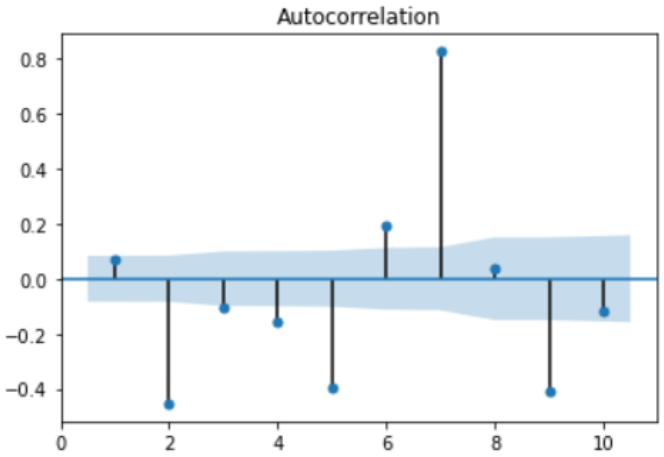
- Bu kısımda öncelikle statsmodels kütüphanesini import ediyoruz ve otokorelasyon ve kısmi oto korelasyon için plot_acf ve plot_pacf'yi import ediyoruz.
- Otokorelasyon grafiği için plot_acf'yi kullanıyoruz ve içerisine train_df'yi yazıp gecikme değerinide 100 yapıyoruz.
- Komutu çalıştırdığımızda otokorelasyon grafiği direk karşımıza çıkacaktır.

```
In [82]: 1 plot_pacf(train_df, lags=10, zero=False);
```



- Kısmi otokorelasyon grafiği için plot_pacf'yi kullanıyoruz ve içerisine train_df'yi yazıp gecikme değerinide 10 yapıyoruz.
- Komutu çalıştırdığımızda kısmi otokorelasyon grafiği direk karşımıza çıkacaktır.

```
In [83]: 1 plot_acf(np.diff(train_df, 1), lags=10, zero=False);
```



- Burdaki bölümdede kısmi otokorelasyon grafiği oluşturuyoruz ama np.diff ile train_df verisi boyunca 1'inci ayrık farkı hesaplamasını sağladık.
- Gecikme süresini 10 yaptık.
- Komutu çalıştırdığımızda kısmi otokorelasyon grafiği direk karşımıza çıkacaktır.

3.2.2 Veri Seti Ön İşleme

```
In [84]:  1  test_df=dataset[554:693]
```

- Dataset(veri setimizdeki) 554 satıra ve 693 satıra kadar olan bölümleri ayırıp test_df'ye eşitliyoruz.

```
In [85]:  1  window_size = 7
          2
          3  numbers_series = pd.Series(train_df)
          4  windows = numbers_series.rolling(window_size)
          5  moving_averages = windows.mean()
          6
          7  moving_averages_list = moving_averages.tolist()
          8  ts7 = moving_averages_list[window_size - 1:]
          9  ts7=np.array(ts7)
         10  print(ts7)
```

- Öncelikle windows_size ile pencere boyunu 7 olarak ayarlıyoruz.
- Pd.Series ile train_df'deki verileri elde tuttuk ve numbers_series'e eşitledik.
- Rolling metodu kullanarak numbers_series'i pencere boyutu 7 olan hareketli pencereyi hesapladık ve sonucu windows'a eşitledik.
- Windows.mean ile aritmetik ortalamasını hesapladık ve moving_averages'e eşitledik.
- Moving_averages.tolist ile moving_averages içindeki listeyi moving_averages_list'e eşitledik.
- moving_averages_list üzerinde satır kısmından windows_size – 1 işlemi gerçekleştirildi. Sütun aynı kaldı ve bunuda ts7'ye eşitledik.
- 'ts7''yi np.array ile Numpy Array'e Dönüştürdük.
- Print ile ts7'yi ekrana yazdırdık.
- Çıktı aşağıda doğru devam etmektedir.

Çıktısı;

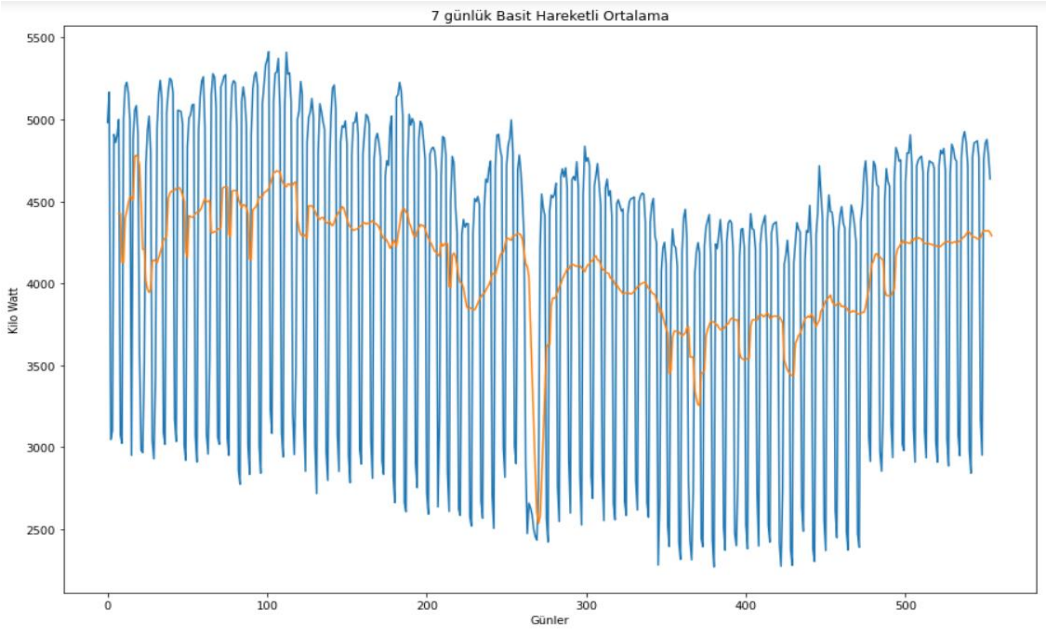
```
[4423.95016863 4426.58588328 4127.91444996 4124.65730693 4396.58588136
 4438.11445782 4490.72159985 4527.51446069 4526.31445775 4508.35731098
 4774.11445863 4781.335889   4764.87874848 4720.20018032 4494.34302684
 4207.20015744 4209.60015842 4014.15015384 3966.42872428 3944.67872128
 3959.82871954 4133.8072943  4142.61444282 4137.19301604 4124.01444408
 4156.65015984 4188.53586798 4219.77872932 4266.87872752 4273.41444343
 4286.03586934 4517.25016077 4548.85730309 4562.55017281 4562.63588824
 4567.56446431 4579.09303531 4581.55732456 4583.57161442 4566.68590411
 4537.73589653 4499.50732123 4192.30731256 4156.56445257 4413.04303441
 4406.16445869 4402.65017155 4408.41445515 4425.64302173 4433.74301965
 4432.37159022 4445.74302211 4464.15017019 4493.52874484 4518.06446212
 4497.96446582 4499.01446778 4505.91446904 4307.50731386 4309.05016518
 4315.50016376 4315.28587505 4332.77158957 4330.62873243 4339.17873411
 4576.86445837 4588.15731976 4586.2287516  4588.20018196 4300.00731877
 4284.02160181 4563.8144662  4565.01446479 4566.3858975  4560.25731933
 4513.97160558 4485.72874207 4460.44302506 4479.25731143 4477.97160013
 4461.70731408 4424.25016894 4146.83587074 4143.3430132  4448.01445524
 4458.68588556 4468.24302864 4491.94303022 4526.27160481 4526.10017531
 4527.08589009 4554.06445693 4557.70731162 4567.22158976 4578.04301698
 4608.493021   4640.16445569 4674.85731644 4676.97875515 4687.17876217
```

In [86]:

```
n1=math.nan
n7=np.array([n1,n1,n1,n1,n1,n1,n1])
ts7=np.concatenate([n7,ts7])
plt.figure(figsize=(15,10))
plt.plot(train_df)
plt.plot(ts7)
plt.xlabel('Günler')
plt.ylabel('Kilo Watt')
plt.title("7 günlük Basit Hareketli Ortalama")
plt.show()
```

- Math.nan kayan nokta nan (Sayı) değerini verdiği için kullandık ve n1'e eşitledik.
- Dizi şeklinde n1'lerde 7 tane yazdık ve n7'ye eşitledik.
- Concatenate ile 2 diziyi birleştirdik bu diziler n7 ve ts7.
- Daha sonra figsize ile grafiğin en boy oranını belirledik.
- Plt.plot ile veri setimiz olan train_df'yi içine yazıyoruz.
- Plt.plot ile ayarladığımız veri seti olan ts7'yi içine yazıyoruz.
- X ekseni üzerine etiket atayarak 'Günler' yazısı yazdırıyoruz.
- Y ekseni üzerine etiket atayarak 'Kilo Watt' yazısı yazdırıyoruz.
- Grafiğe başlık olarakta title ile '7 günlük Basit Hareketli Ortalama' yazıyoruz.
- Plt.show ile ekrana gösteriyoruz.

Çıktısı;



```
In [87]:
1 import statsmodels.tsa.stattools as ts
2 #Dickey-Fuller Test
3 result = ts.adfuller(train_df, 1)
4 result
```

```
Out[87]: (-20.93887846541725,
          0.0,
          1,
          552,
          {'1%': -3.4422521197633187,
           '5%': -2.866790184232015,
           '10%': -2.569566175304558},
          8875.183115493921)
```

- Öncelikle statsmodel.tsa.stattools kütüphanesini import ediyoruz.
- Train_df veri setinde Dickey_Fuller Testi yapıyoruz.
- Adfuller'ı veri setimizin sabit olup olmadığını test etmek için kullanıyoruz ve result'a eşitliyoruz.
- Result diyerek ekrana yazdırıyoruz. Yukarıda çıktısı bulunmaktadır.

3.2.3 ARIMA Model Oluşturma

```
In [160]:
1 import pmdarima as pm
2 Arima_model=pm.auto_arima(train_df, start_p=0, start_q=0, max_p=10, max_q=10, start_P=0, start_Q=0, max_P=10, max_Q=10,
3                           m=14, stepwise=True, seasonal=True, information_criterion='aic', trace=True, d=1, D=1,
4                           error_action='warn', suppress_warnings=True, random_state = 20, n_fits=30)
```

- Öncelikle pmdarima kütüphanesini import ediyoruz.
- Pm.auto_arima ile mevsimsel otomatik arima modelini oluşturuyoruz.
- Train_df ile veri setimizi belirleyip start p ve q ile max p ve q'yu yazıyoruz.
- Daha sonra start P ve Q ile max P ve Q'yu yazıyoruz ve m değerini 14 yapıyoruz.
- Daha sonra d'nin değerini ve D'nin değerlerini yazıyoruz.
- Önemli olan burda p,d,q,P,D,Q ve m'dir.

Çıktısı;

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[14]                : AIC=8610.763, Time=0.03 sec
 ARIMA(1,1,0)(1,1,0)[14]                : AIC=8548.188, Time=0.45 sec
 ARIMA(0,1,1)(0,1,1)[14]                : AIC=inf, Time=1.02 sec
 ARIMA(1,1,0)(0,1,0)[14]                : AIC=8579.406, Time=0.04 sec
 ARIMA(1,1,0)(2,1,0)[14]                : AIC=8548.391, Time=1.30 sec
 ARIMA(1,1,0)(1,1,1)[14]                : AIC=8542.359, Time=1.24 sec
 ARIMA(1,1,0)(0,1,1)[14]                : AIC=8550.962, Time=0.24 sec
 ARIMA(1,1,0)(2,1,1)[14]                : AIC=8544.029, Time=4.03 sec
 ARIMA(1,1,0)(1,1,2)[14]                : AIC=8543.749, Time=3.41 sec
 ARIMA(1,1,0)(0,1,2)[14]                : AIC=8552.812, Time=0.58 sec
 ARIMA(1,1,0)(2,1,2)[14]                : AIC=inf, Time=4.60 sec
 ARIMA(0,1,0)(1,1,1)[14]                : AIC=8565.702, Time=1.00 sec
 ARIMA(2,1,0)(1,1,1)[14]                : AIC=8417.212, Time=0.67 sec
 ARIMA(2,1,0)(0,1,1)[14]                : AIC=8421.929, Time=0.30 sec
 ARIMA(2,1,0)(1,1,0)[14]                : AIC=8417.833, Time=0.29 sec
 ARIMA(2,1,0)(2,1,1)[14]                : AIC=8418.488, Time=2.21 sec
 ARIMA(2,1,0)(1,1,2)[14]                : AIC=8417.982, Time=1.81 sec
 ARIMA(2,1,0)(0,1,0)[14]                : AIC=8458.699, Time=0.07 sec
 ARIMA(2,1,0)(0,1,2)[14]                : AIC=8422.462, Time=1.31 sec
 ARIMA(2,1,0)(2,1,0)[14]                : AIC=8418.726, Time=0.56 sec
 ARIMA(2,1,0)(2,1,2)[14]                : AIC=inf, Time=9.24 sec
 ARIMA(3,1,0)(1,1,1)[14]                : AIC=8418.012, Time=0.75 sec
 ARIMA(2,1,1)(1,1,1)[14]                : AIC=inf, Time=3.48 sec
 ARIMA(1,1,1)(1,1,1)[14]                : AIC=inf, Time=2.34 sec
 ARIMA(3,1,1)(1,1,1)[14]                : AIC=inf, Time=3.10 sec
 ARIMA(2,1,0)(1,1,1)[14] intercept      : AIC=8419.214, Time=1.25 sec

Best model:  ARIMA(2,1,0)(1,1,1)[14]
Total fit time: 45.314 seconds
```

```
In [161]:  1  Arima_model.summary()
```

Out[161]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	554
Model:	SARIMAX(2, 1, 0)x(1, 1, [1], 14)	**Log Likelihood**	-4203.606
Date:	Sun, 17 Jan 2021	**AIC**	8417.212
Time:	05:44:08	**BIC**	8438.660
Sample:	0	**HQIC**	8425.601
	- 554		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.3042	0.029	-10.316	0.000	-0.362	-0.246
ar.L2	-0.4659	0.028	-16.722	0.000	-0.520	-0.411
ar.S.L14	-0.5794	0.106	-5.475	0.000	-0.787	-0.372
ma.S.L14	0.3246	0.126	2.573	0.010	0.077	0.572
sigma2	3.556e+05	1.35e+04	26.408	0.000	3.29e+05	3.82e+05

Ljung-Box (L1) (Q):	0.59	**Jarque-Bera (JB):**	396.52
Prob(Q):	0.44	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.14	**Skew:**	-0.08
Prob(H) (two-sided):	0.00	**Kurtosis:**	7.20

- Arima_model.summary() metodu ile model hakkında bilgi özeti ekrana sağlamış olduk.
- ‘.summary()’ özet tablo ve metni ekrana göstermek için kullanılır.

```
In [162]:
1 ARIMAtahmin=pd.DataFrame(Arima_model.predict(n_periods=126), index=test_df)
2 ARIMAtahmin=np.array(ARIMAtahmin)
```

- Arima_model üzerinde predict ile ayarlama yapıp DataFrame ile oluşan modeli tabloya aktardık ve ARIMAtahmin'e eşitledik.
- ARIMAtahmin'i np.array ile diziye dönüştürdük.

```
In [163]:
1 ARIMAtahmin

Out[163]: array([[2942.42081611],
                 [2792.09376583],
                 [4689.7510844 ],
                 [4806.08145043],
                 [4803.01290926],
                 [4789.09551541],
                 [4705.50210819],
                 [3094.35737648],
                 [2893.39718592],
                 [4701.3769366 ],
                 [4805.94869387],
                 [4839.26569607],
                 [4740.10590913],
                 [4598.27014269],
                 [2887.57374771],
                 [2740.95129399],
                 [4653.05248147],
                 [4754.60835184],
                 [4757.62669741],
                 [4754.40638991]
```

- ARIMAtahmin yazarak diziyi ekrana yansıttık.
- Aşağıya doğru dizi devam etmektedir.

```
In [164]:
1 ARIMAtahmin=ARIMAtahmin.reshape(126,-1)
2 ARIMAtahmin

Out[164]: array([[2942.42081611],
                 [2792.09376583],
                 [4689.7510844 ],
                 [4806.08145043],
                 [4803.01290926],
                 [4789.09551541],
                 [4705.50210819],
                 [3094.35737648],
                 [2893.39718592],
                 [4701.3769366 ],
                 [4805.94869387],
                 [4839.26569607],
                 [4740.10590913],
                 [4598.27014269],
                 [2887.57374771],
                 [2740.95129399],
                 [4653.05248147],
                 [4754.60835184],
                 [4757.62669741],
                 [4754.40638991]
```

- Reshape metodu ile ARIMAtahmini yeniden şekillendirdik ve ekrana tekrar tazdırdık.

```
In [165]:  1 len(ARIMAtahmin)

Out[165]: 126
```

- ‘len’ ile ARIMAtahmin’deki öğe sayısını döndürdü ve kaç öğe olduğunu çıktı olarak verdi.

```
In [166]:  1 mse = mean_squared_error(test_df, ARIMAtahmin)
           2 rmse = math.sqrt(mse)
           3 print('RMSE: %f' % rmse)

RMSE: 1280.505178
```

- Arima ve test_df arasında mse değeri hesaplandı.
- Math.sqrt ile mse değerinin karekökü alındı.
- Print ile ‘RMSE:’ yazdırılıp rmse sonucu yazdırıldı.

```
In [167]:  1 np.mean(test_df)

Out[167]: 3862.4563492063494
```

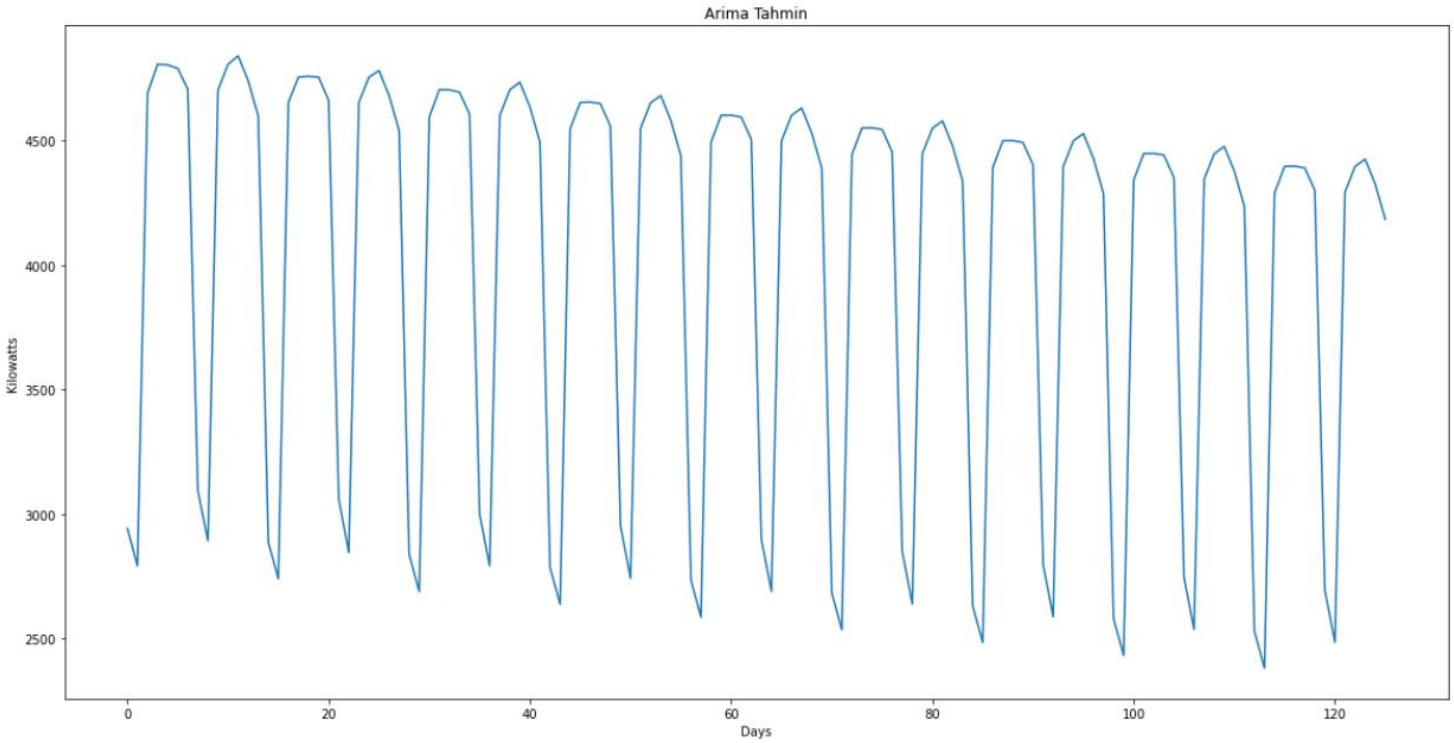
- Np.mean metodu ile test_df’nin aritmetik ortalaması hesaplandı ve ekrana yazdırıldı.

3.2.4 ARIMA Tahmin Grafiği

In [168]:
```
plt.figure(figsize=(20,10))
plt.plot(ARIMAtahmin)
plt.xlabel('Days')
plt.ylabel('Kilowatts')
plt.title("Arima Tahmin")

plt.show()
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plt.plot içerisine ARIMAtahmin modelimizi yazıyoruz.
- X ekseni üzerinde bir etiket oluşturarak ‘Days’ yazdırıyoruz.
- Y ekseni üzerinde bir etiket oluşturarak ‘Kilowatts’ yazdırıyoruz.
- Grafiğe başlık olarak title ile ‘Arima Tahmin’ yazdırıyoruz.
- Plt.show ile ekrana gösteriyoruz.

Çıktısı;

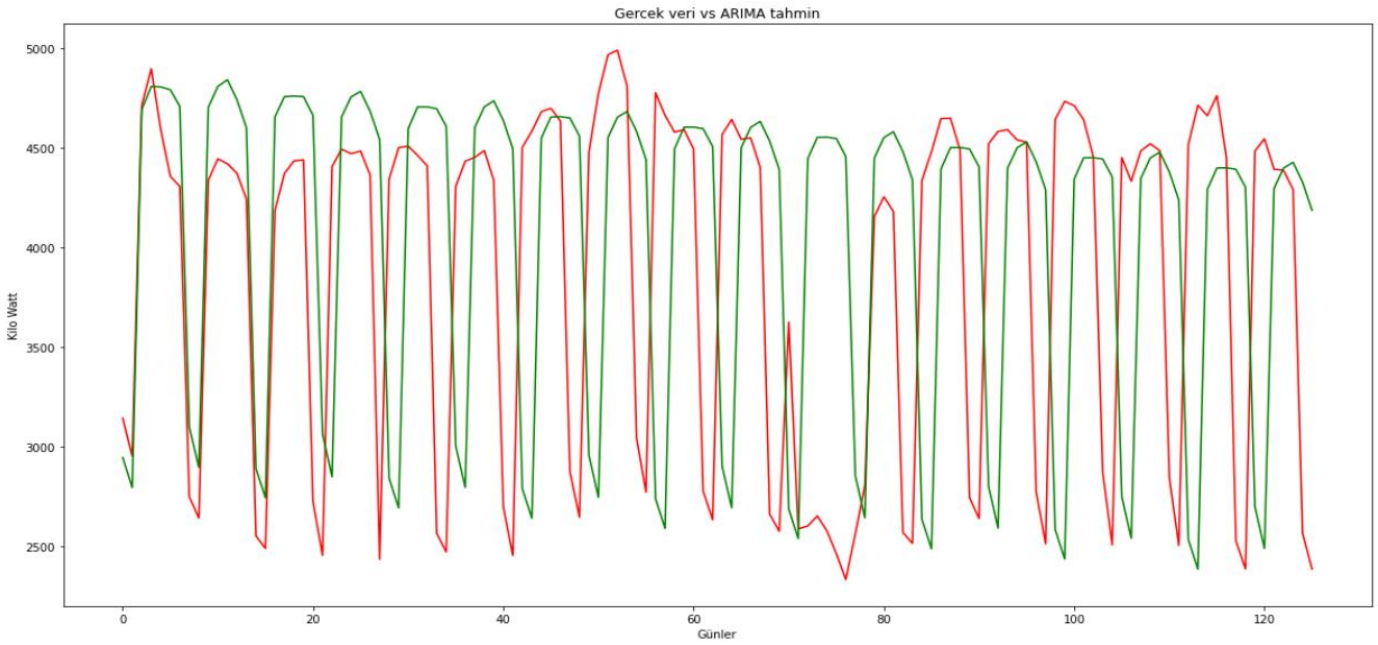


3.2.5 Gerçek Veri ile ARIMA Tahmin Karşılaştırma Grafiği

```
In [169]:
1 plt.figure(figsize=(20,10))
2 plt.plot(test_df, color="red")
3 plt.plot(ARIMAtahmin, color="green")
4 plt.xlabel('Günler')
5 plt.ylabel('Kilo Watt')
6 plt.title("Gercek veri vs ARIMA tahmin")
7 plt.show()
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plt.plot içerisine test_df modelimizi yazıyoruz ve color ile rengini kırmızı yapıyoruz.
- Plt.plot içerisine ARIMAtahmin modelimizi yazıyoruz ve color ile rengini yeşil yapıyoruz.
- X ekseni üzerinde bir etiket oluşturarak ‘Günler’ yazdırıyoruz.
- Y ekseni üzerinde bir etiket oluşturarak ‘Kilo Watt’ yazdırıyoruz.
- Grafiğe başlık olarak title ile ‘Gercek veri vs ARIMA tahmin’ yazdırıyoruz.
- Plt.show ile ekrana gösteriyoruz.

Çıktısı;



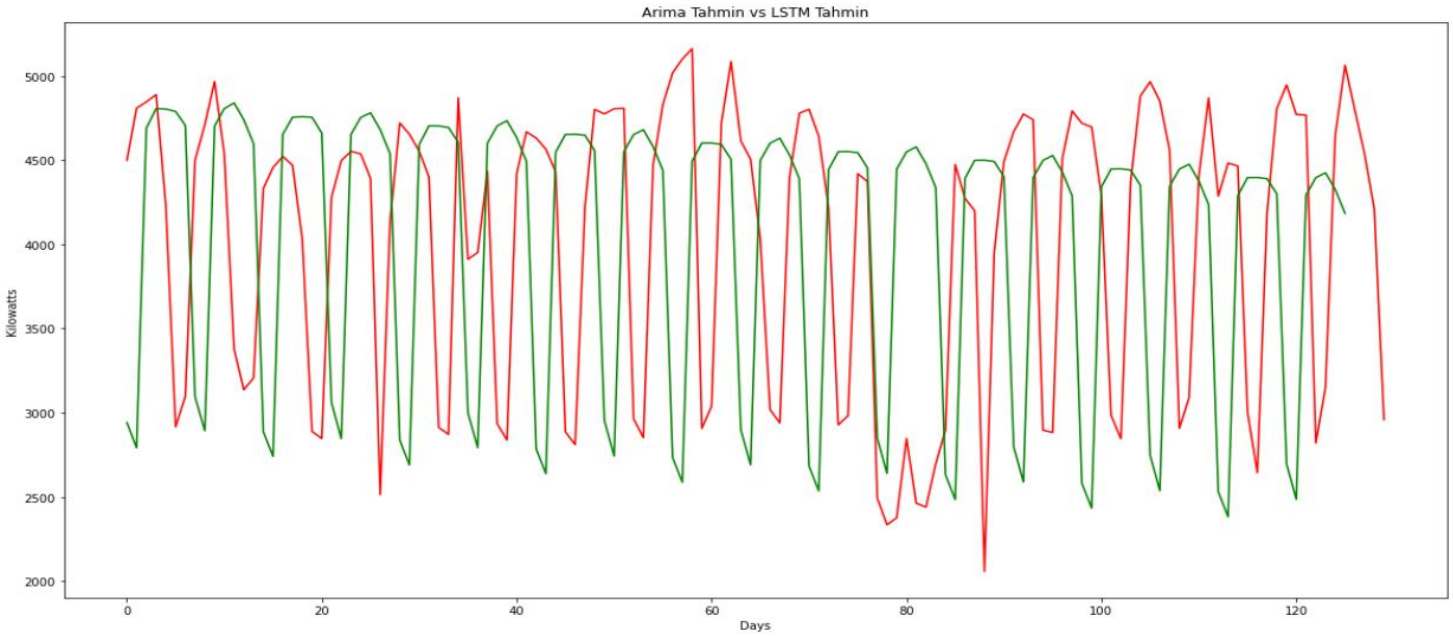
3.2.6 LSTM Tahmin ile ARIMA Tahmin Modeli Karşılaştırma Grafiği

LSTM vs ARİMA

```
In [170]:
1  plt.figure(figsize=(20,10))
2  plt.plot(LSTMtahmin, color="red")
3  plt.plot(ARIMAtahmin, color="green")
4  plt.xlabel('Days')
5  plt.ylabel('Kilowatts')
6  plt.title("Arima Tahmin vs LSTM Tahmin")
7  plt.show()
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plt.plot içerisine LSTMtahmin modelimizi yazıyoruz ve color ile rengini kırmızı yapıyoruz.
- Plt.plot içerisine ARIMAtahmin modelimizi yazıyoruz ve color ile rengini yeşil yapıyoruz.
- X ekseni üzerinde bir etiket oluşturarak ‘Days’ yazdırıyoruz.
- Y ekseni üzerinde bir etiket oluşturarak ‘Kilowatts’ yazdırıyoruz.
- Grafiğe başlık olarak title ile ‘ARIMA Tahmin vs LSTM Tahmin’ yazdırıyoruz.
- Plt.show ile ekrana gösteriyoruz.

Çıktısı;

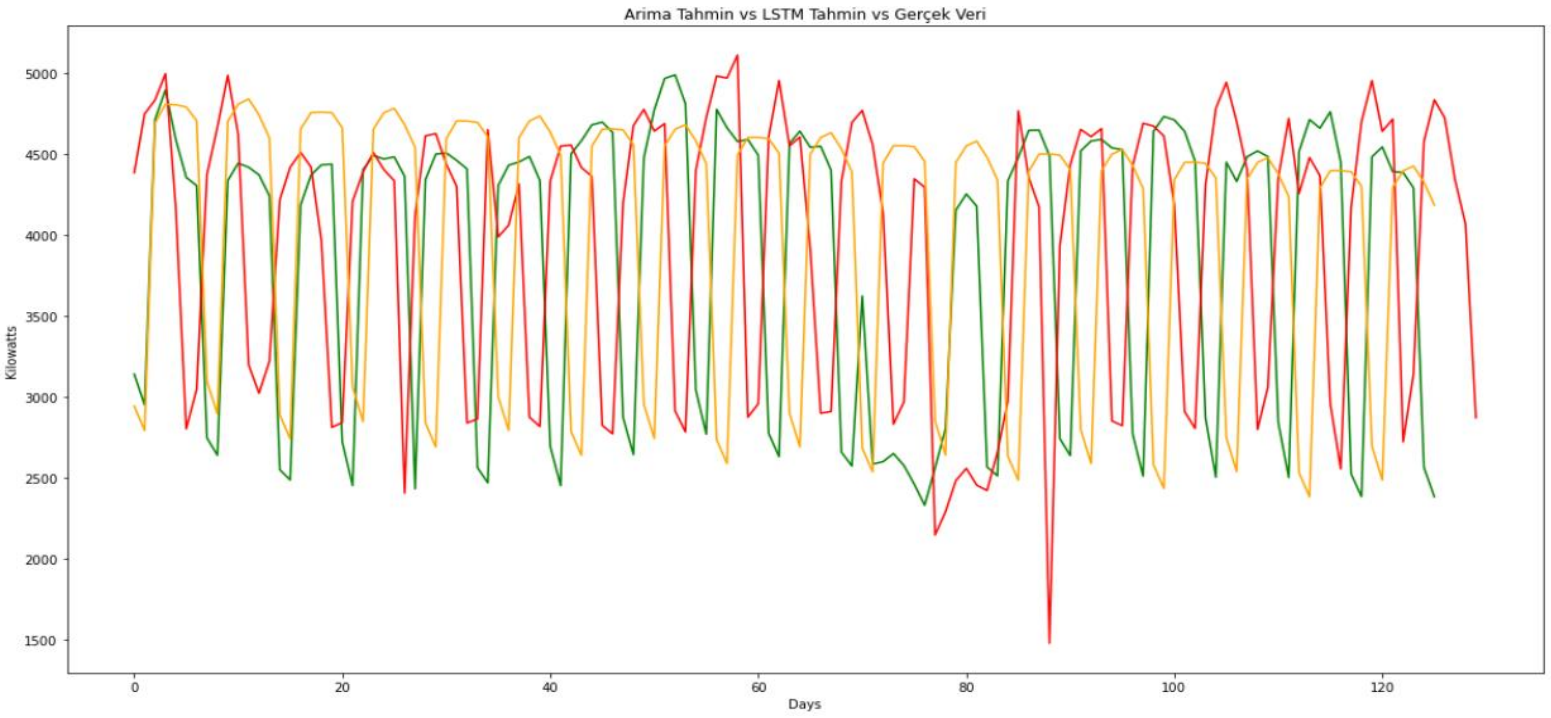


3.2.7 ARIMA Tahmin, LSTM Tahmin ve Gerçek Veri Karşılaştırma Grafiği

```
In [243]:
1  plt.figure(figsize=(20,10))
2  plt.plot(test_df, color="green")
3  plt.plot(LSTMtahmin, color="red")
4  plt.plot(ARIMAtahmin, color="orange")
5  plt.xlabel('Days')
6  plt.ylabel('Kilowatts')
7  plt.title("Arima Tahmin vs LSTM Tahmin vs Gerçek Veri")
8  plt.show()
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plt.plot içerisine test_df modelimizi yazıyoruz ve color ile rengini yeşil yapıyoruz.
- Plt.plot içerisine LSTMtahmin modelimizi yazıyoruz ve color ile rengini kırmızı yapıyoruz.
- Plt.plot içerisine ARIMAtahmin modelimizi yazıyoruz ve color ile rengini turuncu yapıyoruz.
- X ekseni üzerinde bir etiket oluşturarak 'Days' yazdırıyoruz.
- Y ekseni üzerinde bir etiket oluşturarak 'Kilowatts' yazdırıyoruz.
- Grafiğe başlık olarak title ile 'Arima Tahmin vs LSTM Tahmin vs Gerçek Veri' yazdırıyoruz.
- Plt.show ile ekrana gösteriyoruz.

Çıktısı;



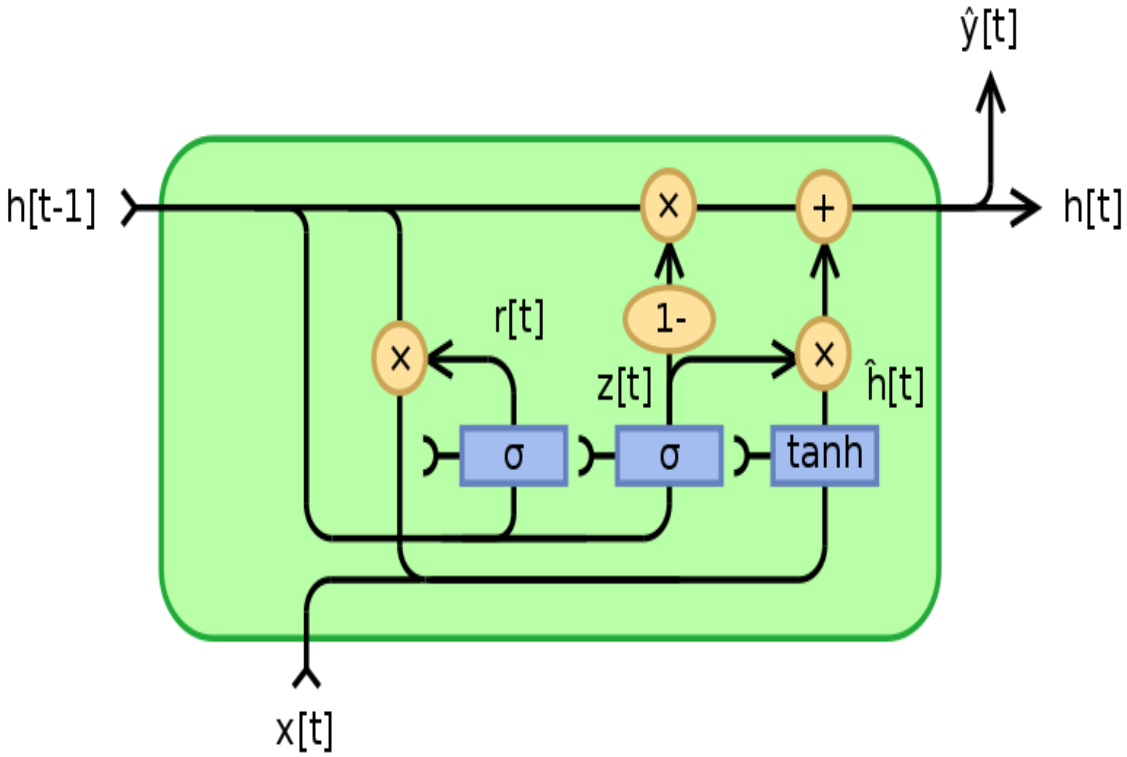
3.3 Multivariable(Çok Değişkenli) GRU Model

Geçitli Tekrarlayan Birim (GRU), RNN mimarisinin bir varyantıdır ve sinir ağındaki hücreler arasındaki bilgi akışını kontrol etmek ve yönetmek için geçit mekanizmaları kullanır. GRU'lar yalnızca 2014 yılında Cho ve diğerleri tarafından tanıtıldı . ve özellikle 1997 yılında Sepp Hochreiter ve Jürgen Schmidhuber tarafından önerilen, yaygın olarak benimsenen LSTM ile karşılaştırıldığında nispeten yeni bir mimari olarak kabul edilebilir .

GRU'nun yapısı, dizinin önceki bölümlerindeki bilgileri atmadan büyük veri dizilerinden bağımlılıkları uyarlamalı olarak yakalamasına olanak tanır. Bu, geleneksel RNN'lerin kaybolan / patlayan gradyan problemini çözen, LSTM'lerdekine benzer , geçitleme birimleri aracılığıyla elde edilir . Bu kapılar, her adımda saklanacak veya atılacak bilgilerin düzenlenmesinden sorumludur.

Dahili geçit mekanizmalarının dışında, GRU, sıralı giriş verilerinin bellekle birlikte her adımda GRU hücresi tarafından tüketildiği veya başka bir şekilde gizli durum olarak bilindiği bir RNN gibi işlev görür . Gizli durum daha sonra dizideki bir sonraki girdi verisi ile birlikte RNN hücresine yeniden beslenir. Bu işlem bir röle sistemi gibi devam ederek istenilen çıktıyı üretir.

Örnek bir LSTM modeli aşağıdaki gibi çalışır.



3.3.1 Kütüphaneler ve Veri seti İşlemleri

In [1]:
```
import numpy as np
np.random.seed(1)
import tensorflow
tensorflow.random.set_seed(2)
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from keras.models import Sequential, load_model
from keras.layers.core import Dense
from keras.layers.recurrent import GRU
from keras import optimizers
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
from math import sqrt
import datetime as dt
import time
plt.style.use('ggplot')
```

- Öncelikle gerekli kütüphaneleri import ediyoruz ve as ile kısaltıyoruz. İmport ettiğimiz kütüphaneler sırasıyla; numpy, pandas, tensorflow, seaborn, matplotlib, keras, sklearn şeklinde devam ediyor.

Veri Ön İşleme

In [2]:
```
# Erken durdurma kurma
earlystop = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=80,  verbose=1, mode='min')
callbacks_list = [earlystop]
```

- Öncelikle earystop(Erken durdurma) metodunu uyguluyoruz.
- Bu işlemi eğer Gru modelin doğrulama veri kümesindeki performansı azalmaya başlarsa bu sayede earlystop ile erken durduma işlemi yapılacak.
- Daha sonra earlystop'u callbacks_list'e eşitliyoruz.

In [3]:
```
# Veri kümesini yükleme
df=pd.read_csv("household_daily.csv",header=0,infer_datetime_format=True)

df.head()
```

Out[3]:

	datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Sub_metering_4
0	2006-12-16	1209.176	34.922	93552.53	5180.8	0.0	546.0	4926.0	14680.933319
1	2006-12-17	3390.460	226.006	345725.32	14398.6	2033.0	4187.0	13341.0	36946.666732
2	2006-12-18	2203.826	161.792	347373.64	9247.2	1063.0	2621.0	14018.0	19028.433281
3	2006-12-19	1666.194	150.942	348479.01	7094.0	839.0	7602.0	6197.0	13131.900043
4	2006-12-20	2225.748	160.998	348923.61	9313.0	0.0	2648.0	14063.0	20384.800011

- Pd.read_csv ile csv dosyasının adını tanımladık(house_daily.csv) ve df'ye eşitledik.
- Df.head metodu ile veri setinin ilk 5 satırını ekrana yazdırdık

Tarih Saat Sütunlarını Yeniden Biçimlendirelim

In [4]:
```
# Yılın Ayı, Günü, Saati vb. Gibi Tüm Verileri Çıkartıyoruz

df["Ay"] = pd.to_datetime(df["datetime"]).dt.month
df["Yil"] = pd.to_datetime(df["datetime"]).dt.year
df["Tarih"] = pd.to_datetime(df["datetime"]).dt.date
df["Saat"] = pd.to_datetime(df["datetime"]).dt.time
df["Hafta"] = pd.to_datetime(df["datetime"]).dt.week
df["Gün"] = pd.to_datetime(df["datetime"]).dt.day_name()
df = df.set_index("datetime")
df.index = pd.to_datetime(df.index)
df.head(5)
```

- Öncelikle veri setimiziin kısaltması olan df yi dataset'e eşitliyoruz.
- dataset içerisine başlığımızı yazıp(Ay) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'mont' yani ay verilerini buraya yazmasını istedik.

- dataset içerisine başlığımızı yazıp(Yil) hangi türden veri çekeceğini yazıyoruz burdaki işlemde ‘year’ yani yıl verilerini buraya yazmasını istedik.
- dataset içerisine başlığımızı yazıp(Tarih) hangi türden veri çekeceğini yazıyoruz burdaki işlemde ‘date’ yani tarih verilerini buraya yazmasını istedik.
- dataset içerisine başlığımızı yazıp(Saat) hangi türden veri çekeceğini yazıyoruz burdaki işlemde ‘time’ yani saat verilerini buraya yazmasını istedik.
- dataset içerisine başlığımızı yazıp(Hafta) hangi türden veri çekeceğini yazıyoruz burdaki işlemde ‘week’ yani hafta verilerini buraya yazmasını istedik.
- dataset içerisine başlığımızı yazıp(Gün) hangi türden veri çekeceğini yazıyoruz burdaki işlemde ‘day_name()’ yani gün verilerini buraya yazmasını istedik.
- set_index() işlevi, var olan sütunları kullanarak veri çerçevesi dizinini ayarlamak için kullanılır.(“Datetime”) ile bu veri çerçevesine bir satır etiketi(başlık) olarak ayarlandı.
- to_datetime, dize tarih saatini python tarih saat nesnesine dönüştürülmesini sağladı.
- dataset.head(5) ile sadece işlemlerin hepsini yazdırmak yerine sadece ilk 5 satırını ekrana yazdırdık.

Çıktısı;

datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Sub_metering_4	Ay	Y
2006-12-16	1209.176	34.922	93552.53	5180.8	0.0	546.0	4926.0	14680.933319	12	200
2006-12-17	3390.460	226.006	345725.32	14398.6	2033.0	4187.0	13341.0	36946.666732	12	200
2006-12-18	2203.826	161.792	347373.64	9247.2	1063.0	2621.0	14018.0	19028.433281	12	200
2006-12-19	1666.194	150.942	348479.01	7094.0	839.0	7602.0	6197.0	13131.900043	12	200
2006-12-20	2225.748	160.998	348923.61	9313.0	0.0	2648.0	14063.0	20384.800011	12	200

- Sütunlar sağa doğru devam etmektedir.

3.3.2 Grafikler

Grafikler, çeşitli yapay ve doğal süreçler tarafından üretilen zengin ve karmaşık verileri temsil etmek için kullanılan bir araçtır. Bir grafik , düğümlere (bilgiyi tutan varlıklar) ve kenarlara (aynı zamanda bilgiyi tutan düğümler arasındaki bağlantılar) sahip olan ve bu nedenle, ilişkisel bir doğanın yanı sıra bir bileşim niteliğine sahip olan yapılandırılmış bir veri türü olarak düşünülebilir. Grafik, verileri yapılandırmanın bir yoludur, ancak kendi başına bir veri noktası da olabilir. Grafikler, bir Öklid dışı veri türüdür , yani görüntüler, metin ve ses gibi diğer veri türlerinin aksine 3B olarak bulunurlar. Grafikler, üzerlerinde gerçekleştirilebilecek olası eylemleri ve analizleri sınırlayan belirli özelliklere sahip olabilir. Bu bölümde veri setimizi grafiklere dökmeyi göstereceğiz.

Global_active_power'ın Grafikleri

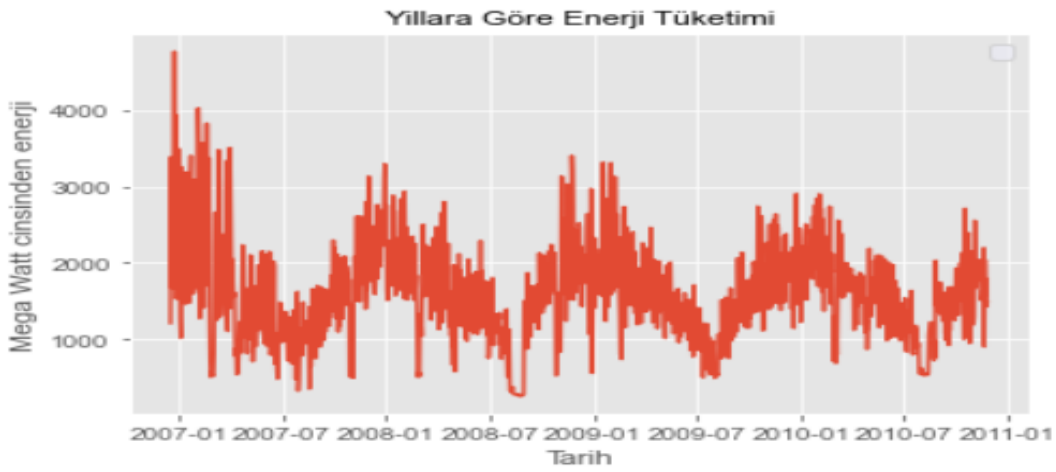
Çizgi Grafiği

Çizgi grafiği, verileri her bir değerin sıklığını gösteren bir sayı doğrusu üzerinde noktalar veya onay işaretleri olarak görüntüleyen bir grafik olarak tanımlanabilir.

In [5]:
```
#Çizgi Grafiği
sns.lineplot(x=df["Tarih"], y=df["Global_active_power"], data=df)
sns.set(rc={'figure.figsize':(15,6)})
plt.xlabel("Tarih")
plt.ylabel("Mega Watt cinsinden enerji")
plt.legend()
plt.title("Yillara Göre Enerji Tüketimi")
```

- sns.lineplot ile seaborn kütüphanesinden çizgi grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setmizden ‘Tarih’ kısmını yazdırdık, y değerine de veri setmizden ‘Global_active_power’ kısmını kullanıp yazdırdık.
- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- Grafiğin x koordinatı altına etiket olarak ‘Tarih’ yazdırdık.
- Grafiği y koordinatı altınada etiket olarak ‘Mega Watt cinsinden enerji’ yazdırdık.
- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- Title metodunu kullanarak grafiğe başlık(Yillara Göre Enerji Tüketimi) yazdırdık.

Çıktısı;



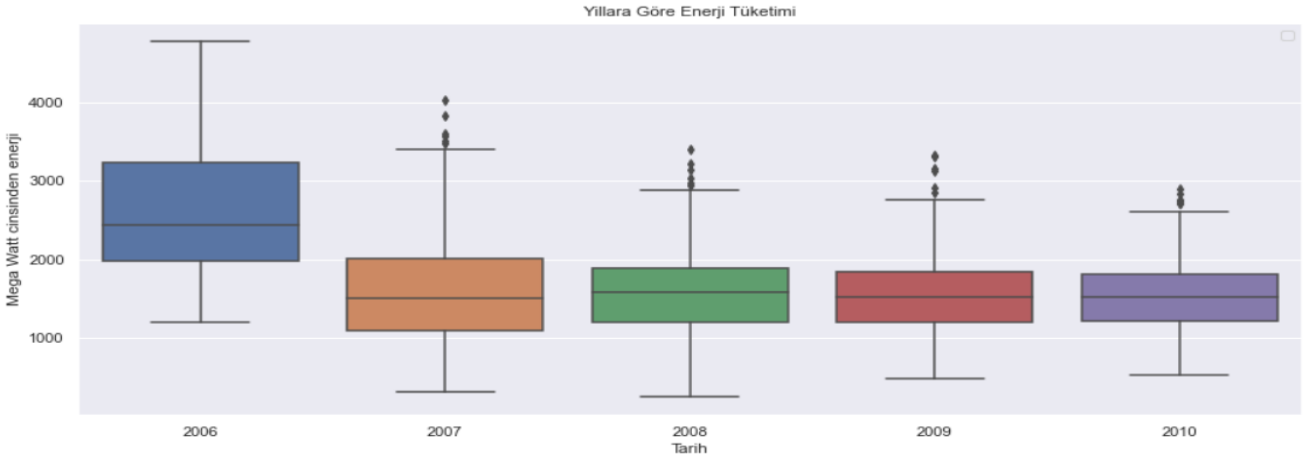
Kutu Grafiği

Kutu grafiği, ilgili değişken bakımından veri için hazırlanan beş sayılı özetleme tablosu gösterimini grafiksel olarak özetlemeye dayalıdır. Özellikle merkezsel konum, yayılma, çarpıklık ve basıklık yönünden verileri özetlemek ve aykırı değerleri tanımlamak için kullanılır.

In [6]:
```
#Kutu Grafiği
sns.boxplot(x=df["Yil"], y=df["Global_active_power"], data=df)
sns.set(rc={'figure.figsize':(15,6)})
plt.xlabel("Tarih")
plt.ylabel("Mega Watt cinsinden enerji")
plt.legend()
plt.title("Yillara Göre Enerji Tüketimi")
```

- sns.boxplot ile seaborn kütüphanesinden kutu grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setmizden 'Yil' kısmını yazdırdık, y koordinatına da veri setmizden 'Global_active_power' kısmını kullanıp yazdırdık.
- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- Grafiğin x koordinatı altına etiket olarak 'Tarih' yazdırdık.
- Grafiği y koordinatı altınada etiket olarak 'Mega Watt cinsinden enerji' yazdırdık.
- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- Son olarakta Grafiğe bir başlık eklemek için Title metodu ile 'Yillara Göre Enerji Tüketimi' yazdırıyoruz.

Çıktısı;



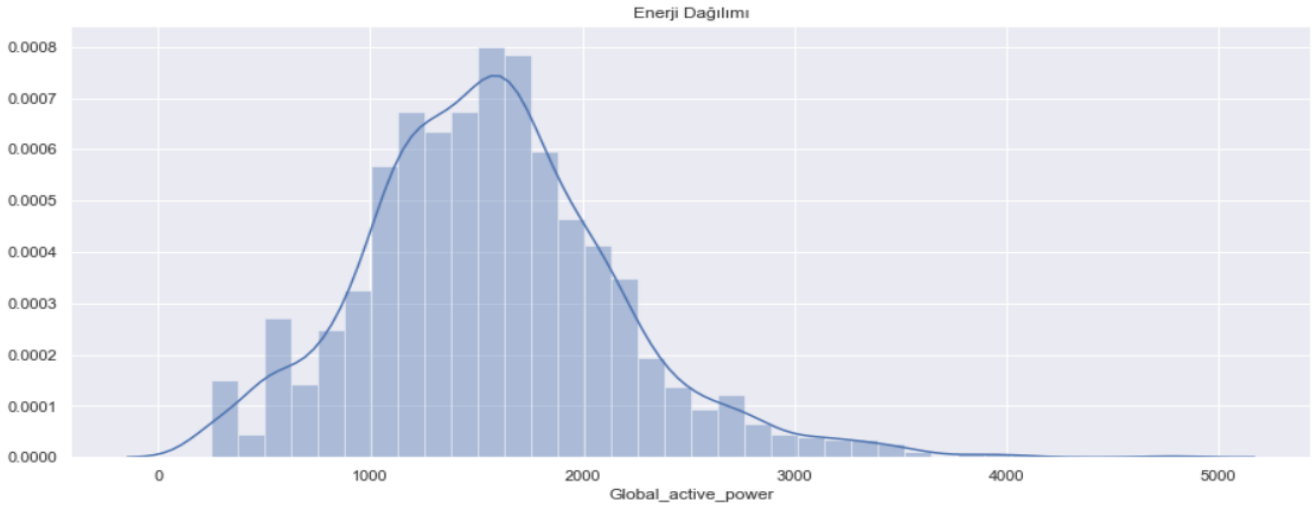
Histogram Grafiği

Çubuk Grafiğine benzer , ancak histogram sayıları aralıklar halinde gruplandırır . Her çubuğun yüksekliği, her bir aralığa kaç tane düştüğünü gösterir. Ve hangi aralıkları kullanacağımıza biz karar veririz.

In [7]:
```
#Histogram Grafiği
sns.distplot(df["Global_active_power"])
plt.title("Enerji Dağılımı")
```

- sns.distplot modülü üzerinde çizgi bulunan bir histogram grafiği göstermemiz sağlar. Bu modül ile veri setindeki Global_active_power değerlerini histogram grafiğine aktarmış olundu.
- Grafiğimize başlık olarak plt.title metodunu kullanıp ‘Enerji Dağılımı’ başlığı attık.

Çıktısı;



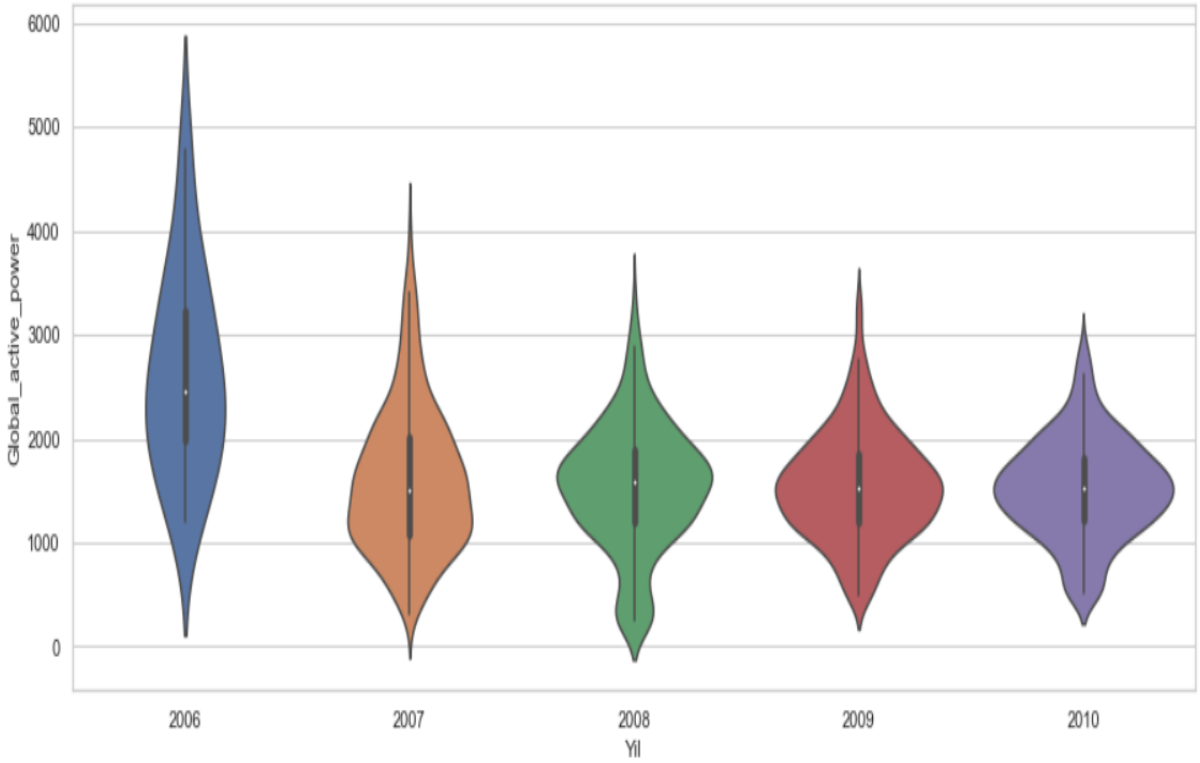
Violin(Keman) Grafiği

Bir keman grafiği , sayısal verileri çizme yöntemidir. Her bir tarafa döndürülmüş bir çekirdek yoğunluğu grafiğinin eklenmesiyle bir kutu grafiğine benzer . Keman grafikleri , genellikle bir çekirdek yoğunluğu tahmincisi tarafından düzleştirilen farklı değerlerde verilerin olasılık yoğunluğunu göstermeleri dışında kutu grafiklerine benzer. Tipik olarak bir keman grafiği, bir kutu grafiğindeki tüm verileri içerir. Bir keman grafiğinin birden fazla katmanı olabilir. Örneğin, dış şekil tüm olası sonuçları temsil eder. İçerideki bir sonraki katman, zamanın% 95'inde oluşan değerleri temsil edebilir. İçerideki bir sonraki katman (varsa), zamanın% 50'sinde oluşan değerleri temsil edebilir.

```
In [8]:
1 #Violin(keman) Grafiği
2 sns.set_style('whitegrid')
3 sns.violinplot(x = 'Yil', y = 'Global_active_power', data = df)
```

- Öncelikle stil setinden arka plan olarak beyaz ızgara(whitegrid) rengini seçtik.
- Sns.violinplot ile seaborn kütüphanesinden violin grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setimizden 'Yil' kısmını kullandık, y koordinatınada veri setimizden 'Global_active_power' kısmını kullanıp yazdırdık.

  Çıktısı;



Global_reactive_power'ın Grafikleri

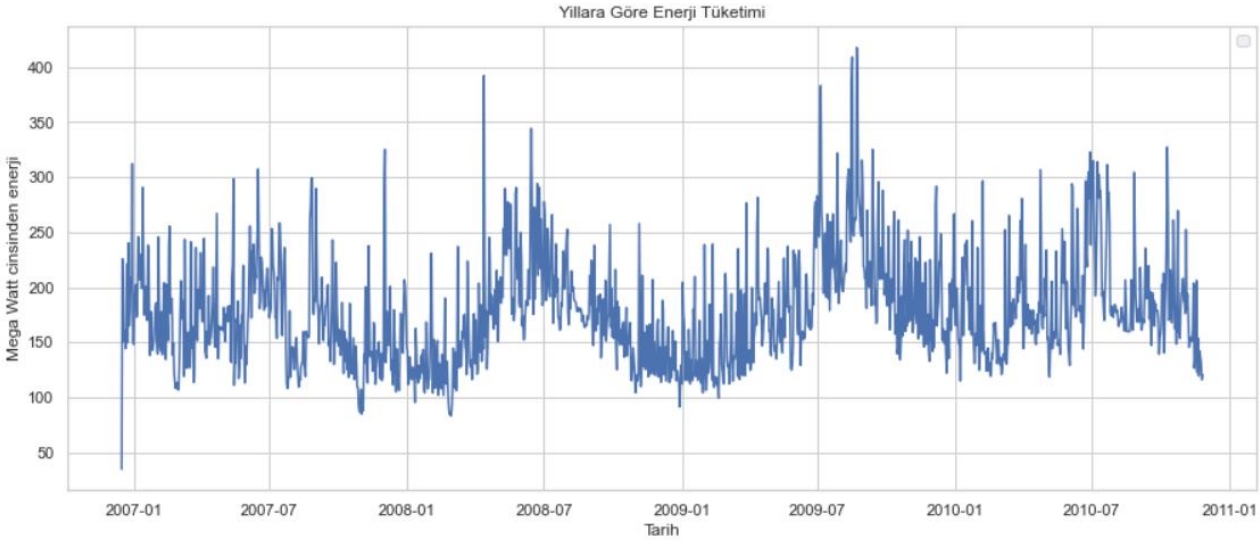
Çizgi Grafiği

In [9]:
```
#Çizgi Grafiği
sns.lineplot(x=df["Tarih"], y=df["Global_reactive_power"], data=df)
sns.set(rc={'figure.figsize':(15,6)})
plt.xlabel("Tarih")
plt.ylabel("Mega Watt cinsinden enerji")
plt.grid(True)
plt.legend()
plt.title("Yillara Göre Enerji Tüketimi")
```

```
No handles with labels found to put in legend.
```

- sns.lineplot ile seaborn kütüphanesinden çizgi grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setmizden 'Tarih' kısmını yazdırdık, y değerine de veri setmizden 'Global_reactive_power' kısmını kullanıp yazdırdık.
- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- Grafiğin x koordinatı altına etiket olarak 'Tarih' yazdırdık.
- Grafiği y koordinatı altınada etiket olarak 'Mega Watt cinsinden enerji' yazdırdık.
- Grafikte ızgara çizgilerinin gözükmesini istediğimiz için 'plt.grid(True)' kısmının içini doğru(true) yaptık.
- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- Title metodunu kullanarak grafiğe başlık(Yillara Göre Enerji Tüketimi) yazdırdık.

  Çıktısı;



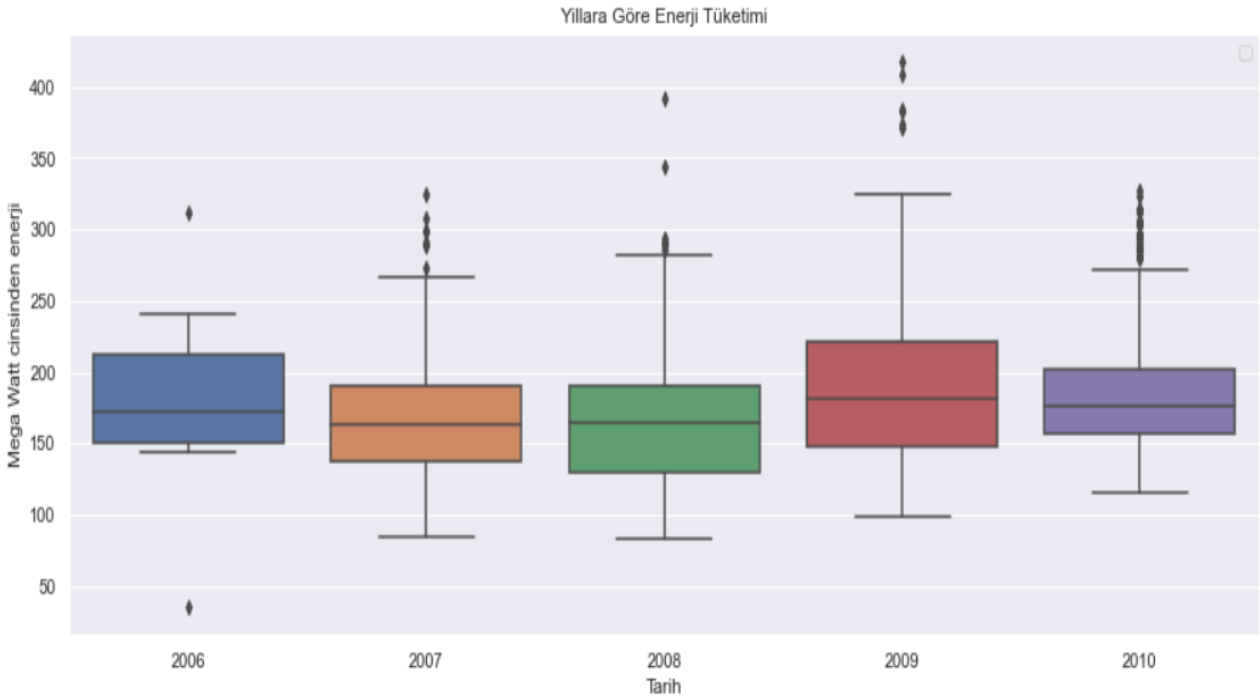
Kutu Grafiği

In [10]:
```
#Kutu Grafiği
sns.boxplot(x=df["Yil"], y=df["Global_reactive_power"], data=df)
sns.set(rc={'figure.figsize':(15,6)})
plt.xlabel("Tarih")
plt.ylabel("Mega Watt cinsinden enerji")
plt.legend()
plt.title("Yillara Göre Enerji Tüketimi")
```

```
No handles with labels found to put in legend.
```

- sns.boxplot ile seaborn kütüphanesinden kutu grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setmizden 'Yil' kısmını yazdırdık, y koordinatına da veri setmizden 'Global_reactive_power' kısmını kullanıp yazdırdık.
- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- Grafiğin x koordinatı altına etiket olarak 'Tarih' yazdırdık.
- Grafiği y koordinatı altınada etiket olarak 'Mega Watt cinsinden enerji' yazdırdık.
- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- Son olarakta Grafiğe bir başlık eklemek için Title metodu ile 'Yillara Göre Enerji Tüketimi' yazdırıyoruz.

  Çıktısı;

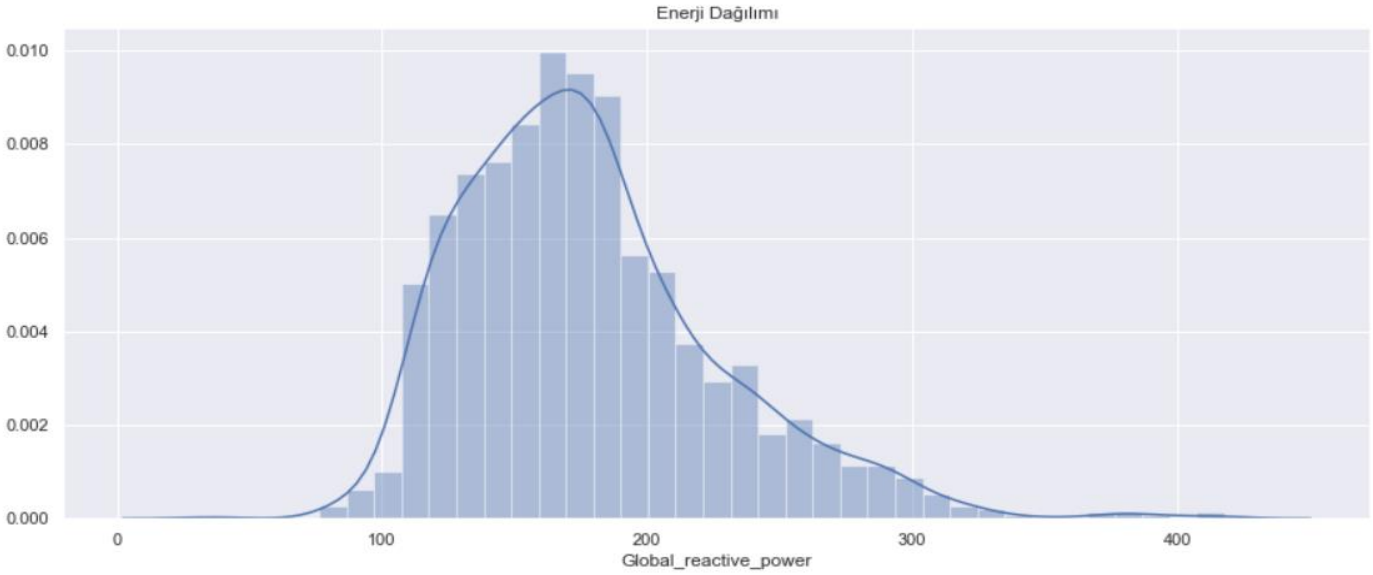


Histogram Grafiği

```
In [11]:
1 #Histogram Grafiği
2 sns.distplot(df["Global_reactive_power"])
3 plt.title("Enerji Dağılımı")
```

- sns.distplot modülü üzerinde çizgi bulunan bir histogram grafiği göstermemiz sağlar. Bu modül ile veri setindeki Global_reactive_power değerlerini histogram grafiğine aktarmış olundu.
- Grafiğimize başlık olarak plt.title metodunu kullanıp 'Enerji Dağılımı' başlığı attık.

  Çıktısı;

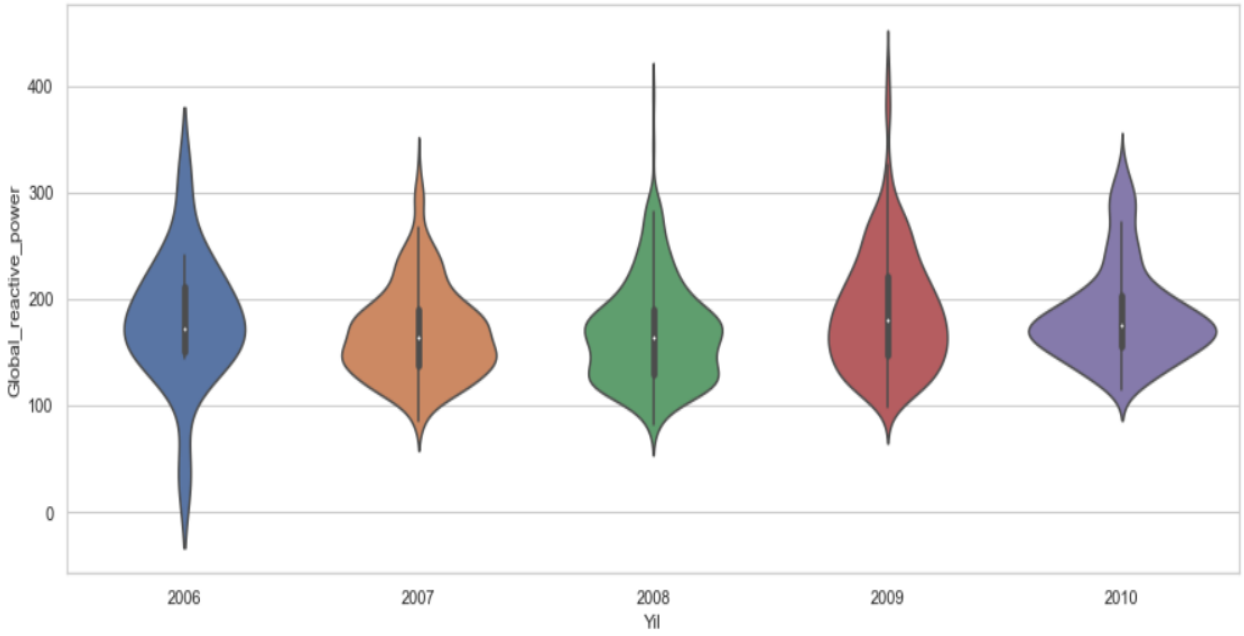


Violin(Keman) Grafiği

```
In [12]:
1 #Violin(keman) Grafiği
2 sns.set_style('whitegrid')
3 sns.violinplot(x = 'Yil', y = 'Global_reactive_power', data = df)
```

- Öncelikle stil setinden arka plan olarak beyaz ızgara(whitegrid) rengini seçtik.
- Sns.violinplot ile seaborn kütüphanesinden violin grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setimizden 'Yil' kısmını kullandık, y koordinatınada veri setimizden 'Global_reactive_power' kısmını kullanıp yazdırdık.

Çıktısı;



3.3.3 GRU Modeli Oluşturma

```
In [13]:  1  # Korelasyon matrisi
          2  df.corr()['Global_active_power']
```

```
Out[13]: Global_active_power      1.000000
         Global_reactive_power    0.042327
         Voltage                  0.065265
         Global_intensity         0.999181
         Sub_metering_1           0.545054
         Sub_metering_2           0.482796
         Sub_metering_3           0.734302
         Sub_metering_4           0.887348
         Ay                      -0.087049
         Yil                     -0.084846
         Hafta                   -0.086835
         Name: Global_active_power, dtype: float64
```

- Veri seti çerçevesindeki tüm sütunların ikili korelasyonunu bulmak için df.corr metodunu kullandık.
- Global_active_power'a göre ikili korelasyon matrisini bulduk ve ekrana yazdırdık.

In [14]:
```
print(df.describe().Global_reactive_power)
df.drop(df[df['Global_reactive_power']==0].index, inplace = True)
#Hacim değeri 0 olan satırları düşürelim
```

```
count    1442.000000
mean      178.004759
std        48.881691
min        34.922000
25%       143.063000
50%       171.199000
75%       202.548500
max       417.834000
Name: Global_reactive_power, dtype: float64
```

- df.describe() metodu sayısal verilere sahip olan sütunların max, min , std…gibi istatiksel değerlerini döndürür.
- Bu yüzden veri setinde olan Global_reactive_power'dan daha çok bilgi edinilmek için kullanıldı ve print ile ekrana yazdırdık.
- Global_reactive_power'da drop metodu ile null(Boş veya sıfır) olan değerleri çıkarttık.
- Çıktısı yukarıdaki gibidir.

In [15]:
```
# Modeli oluşturalım ve eğitelim
def fit_model(train,val,timesteps,hl,lr,batch,epochs):
    X_train = []
    Y_train = []
    X_val = []
    Y_val = []

    # Eğitim verileri için döngü
    for i in range(timesteps,train.shape[0]):
        X_train.append(train[i-timesteps:i])
        Y_train.append(train[i][0])
    X_train,Y_train = np.array(X_train),np.array(Y_train)

    # Val verileri için döngü
    for i in range(timesteps,val.shape[0]):
        X_val.append(val[i-timesteps:i])
        Y_val.append(val[i][0])
    X_val,Y_val = np.array(X_val),np.array(Y_val)

    # Modele Katmanlar Ekleme
    model = Sequential()
    model.add(GRU(X_train.shape[2],input_shape = (X_train.shape[1],X_train.shape[2]),return_sequences = True,
                  activation = 'relu'))
    for i in range(len(hl)-1):
        model.add(GRU(hl[i],activation = 'relu',return_sequences = True))
    model.add(GRU(hl[-1],activation = 'relu'))
    model.add(Dense(1))
    model.compile(optimizer = optimizers.Adam(lr = lr), loss = 'mean_squared_error')

    # Verilerin eğitimi
    history = model.fit(X_train,Y_train,epochs = epochs,batch_size = batch,validation_data = (X_val, Y_val),verbose = 0,
                        shuffle = False, callbacks=callbacks_list)
    model.reset_states()
    return model, history.history['loss'], history.history['val_loss']
```

- Öncelikle X_train, Y_train, X_val ve Y_val oluşturuyoruz.

- Eğitim verilerimiz için döngü oluşturuyoruz ve burda X_train ile Y_traini kullanıp Numpay Array'a dönüştürüp tekrar kendisine eşitliyoruz.
- Eğitim verilerimiz için döngü oluşturuyoruz ve burda X_val ile Y_val kullanıp Numpay Array'a dönüştürüp tekrar kendisine eşitliyoruz.
- Modele katman ekleme işlemi gerçekleştiriyoruz.
- Bu modelde katman olarak gru modeli kullanacağımız için gru modelini yazıyoruz.
- 3 katmanlı bir gru modeli çiziyoruz ve giriş katmanı sayısı 2 ve çıkış katmanı sayısı 1 şeklinde.
- Model.fit ile Verilerin eğitimini gerçekleştiriyoruz .
- X_train, Y_train, X_val ve Y_val gerekli yerlere yazıyoruz ve history'e eşitliyoruz.
- Model.reset.states'i birbirini izleyen model aramaları bağımsız yapmak istediğimizde her seferinde araması için yazdık.
- Modelde elde ettiğimiz loss ve val_loss değerlerini history.history içerisinde belirttik.

In [16]:
```
# Modeli değerlendirme
def evaluate_model(model,test,timesteps):
    X_test = []
    Y_test = []

    # Verileri test etmek için döngü
    for i in range(timesteps,test.shape[0]):
        X_test.append(test[i-timesteps:i])
        Y_test.append(test[i][0])
    X_test,Y_test = np.array(X_test),np.array(Y_test)

    # Tahmin
    Y_hat = model.predict(X_test)
    mse = mean_squared_error(Y_test,Y_hat)
    rmse = sqrt(mse)
    r2 = r2_score(Y_test,Y_hat)
    return mse,rmse, r2, Y_test, Y_hat
```

- Modeli test etmek için X_test ve Y_test oluşturuyoruz.
- Oluşturduğumuz X_test ve Y_test'i verileri test etmesi için döngüye sokuyoruz.
- Np.array ile X_test ile Y_test'i diziye dönüştürdük ve kendisine eşitledik.
- Tahmin değerlerimizi model.predict içerisine X_test şeklinde yazıyoruz ve Y_hat 'a ekliyoruz.

- Y_test ve Y_hat arasında mse değeri hesaplandı.
- Ssqrt ile mse değerinin karekökü alındı ve rmse değeri bulundu.
- R2_score yani belirleme katsayısı, bağımlı değişken içindeki bağımsız değişkenden tahmin edilebilir varyansın oranını hesaplar yani Y_test ile Y_hat arasındaki varyansın oranı hesaplandı.

In [17]:
```
# Eğitim hatalarının grafiğini çizme
def plot_error(train_loss,val_loss):
    plt.plot(train_loss,c = 'r')
    plt.plot(val_loss,c = 'b')
    plt.ylabel('Loss')
    plt.xlabel('Epochs')
    plt.title('Loss Plot')
    plt.legend(['train','val'],loc = 'lower right')
    plt.show()
```

- Plot_error ile güven aralıkları(train_loss ve val_loss) ile çizgi çizeceğiz.
- Plt.plot ile train_loss'u yazdıracağız ve rengini kısaltma r(red) ile kırmızı yapıyoruz.
- Plt.plot ile vall_loss'u yazdıracağız ve rengini kısaltma b(blue) ile mavi yapıyoruz.
- Y ekseni üzerinde bir etiket oluşturuyoruz ve adını 'Loss' koyuyoruz.
- X ekseni üzerinde bir etiket oluşturuyoruz ve adını 'Epochs' koyuyoruz.
- Grafiğe başlık olarak title ile 'Loss Plot' yazıyoruz.
- Plt.legend ile çalıştırıyoruz.
- Plt.show ile ekrana gösteriyoruz.
- Unutmamak gerekirki bu grafiği ileriki kısımlarda tek tek yazmak yerine sadece plot_error yazıp geçeceğiz ve grafiğimiz oluşacak.

In [18]:
```
# Seriyi ekrana çıkarma
series = df[['Global_active_power','Global_reactive_power']] # Özellikleri seçtik.2 Giriş.
print(series.shape)
print(series.tail())
```

```
(1442, 2)
            Global_active_power  Global_reactive_power
datetime
2010-11-22              2041.536                142.354
2010-11-23              1577.536                137.450
2010-11-24              1796.248                132.460
2010-11-25              1431.164                116.128
2010-11-26              1488.104                120.826
```

- Veri setimiz olan df üzerinde Global_active_power ile Global_reactive_power'i seçiyoruz ve series'e eşitliyoruz bu kısmı yapmamızdaki amaç modele 2 giriş sağlamak.
- Series.shape'i print ile ekrana yazdırıyoruz ve kaç tane satır ve sütun verimizin olduğunu ekranda görüyoruz.
- Print kullanmasaydık series.shape gözükmeyecekti.
- Print.series.tail ile tail metodunu kullanarak son 5 veriyi ekrana gösterttik.

In [19]:
```
# Train,Val,Test Bölümü
train_start = dt.date(2006,12,16)
train_end = dt.date(2009,1,24)
train_data = series.loc[train_start:train_end]

val_start = dt.date(2009,1,25)
val_end = dt.date(2010,1,25)
val_data = series.loc[val_start:val_end]

test_start = dt.date(2010,1,26)
test_end = dt.date(2010,11,26)
test_data = series.loc[test_start:test_end]

print(train_data.shape,val_data.shape,test_data.shape)
```

```
(771, 2) (366, 2) (305, 2)
```

- Bu bölümde Train,Val,Test Bölümünün başlangıç(start) ve bitiş(end) bölümlerinin tarihlerini ayırdık.
- Daha sonra print ile train_data.shape , val_data.shape, test_data.shape ile ekrana veri sayısını(sütunları ve satırlarını) gösterttik.

In [20]:
```
# Normalleştirme
sc = MinMaxScaler()
train = sc.fit_transform(train_data)
val = sc.transform(val_data)
test = sc.transform(test_data)
print(train.shape,val.shape,test.shape)
```

```
(771, 2) (366, 2) (305, 2)
```

- MinMaxScaler'i sc'ye eşitliyoruz.
- train_data'nın eğitim setine parametrelerin uydurulması için sc.fit_transform() methodunu kullandık ve train'e eşitledik.
- val_data'nın eğitim setine parametrelerin uydurulması için sc.transform() methodunu kullandık ve val'a eşitledik.
- test_data'nın eğitim setine parametrelerin uydurulması için sc. transform() methodunu kullandık ve test'e eşitledik.
- Son olarak print ile train.shape, val.shape, test.shape ile ekrana veri sayısını(sütunları ve satırlarını) gösterttik.

In [21]:
```
timesteps = 40
hl = [40,35]
lr = 1e-4
batch_size = 16
num_epochs = 200
```

- 'timesiteps' ayarını 40 yapıyoruz.
- 'hl' ayarını 40,35 yapıyoruz.
- 'lr' ayarını 1e-4 yapıyoruz.
- 'Batch_size' ile alt küme boyutunu 16 yapıyoruz.

-'Num_epochs' ile veri kümemizin yalnızca bir kez sinir ağından ileri ve geri aktarılmasıdır. Biz 200 kez ileri geri aktardık.

In [22]:
```
#Bu işlem 5-10 dakika arası sürebilir.
model,train_error,val_error = fit_model(train,val,timesteps,hl,lr,batch_size,num_epochs)
```

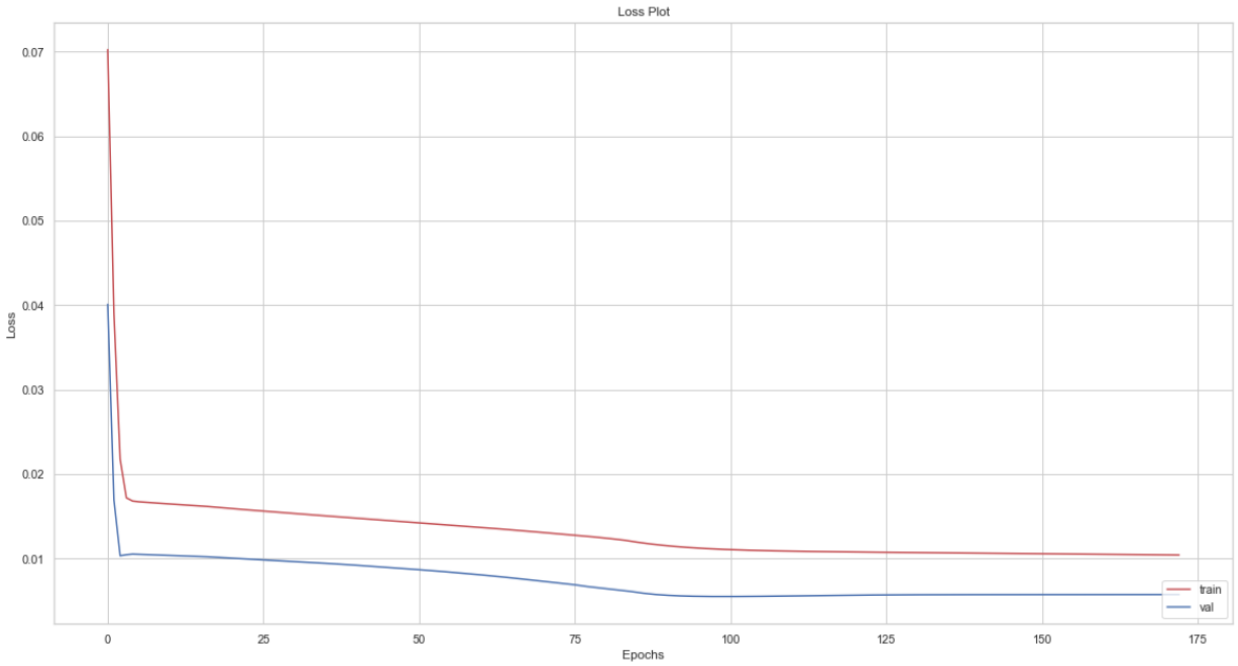
```
Epoch 00173: early stopping
```

- Burdaki işlemde model,train_error,val_error u fit modele eşitledik. Model içerisinede gerekli parametreleri yazdık.

```
In [23]:
1 plt.figure(figsize=(20,10))
2 plot_error(train_error,val_error)
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plot_error ile direk grafiğin çalışmasını sağladık yukarıki kısımda belirttiğimiz kodları burda yazmamıza gerek kalmayacak.

Çıktısı;




```
In [24]:
1 mse,rmse,r2_value,true,predicted = evaluate_model(model,test,40)
2 print("MSE =",mse)
3 print("RMSE =",rmse)
4 print("R2-Score =",r2_value)
```

```
MSE = 0.004761512262480572
RMSE = 0.06900371194711609
R2-Score = 0.4426656189808871
```

- Ms,rmse,r2_value,true,predicted girdileri belirleyerek evaluate_modelde çıktılarını bulmasını sağladık.
- Print ile 'MSE' değerini yazdırmasını istedik.
- Print ile 'RMSE' değerini yazdırmasını istedik.
- Print ile 'R2-Score' değerini yazdırmasını istedik.

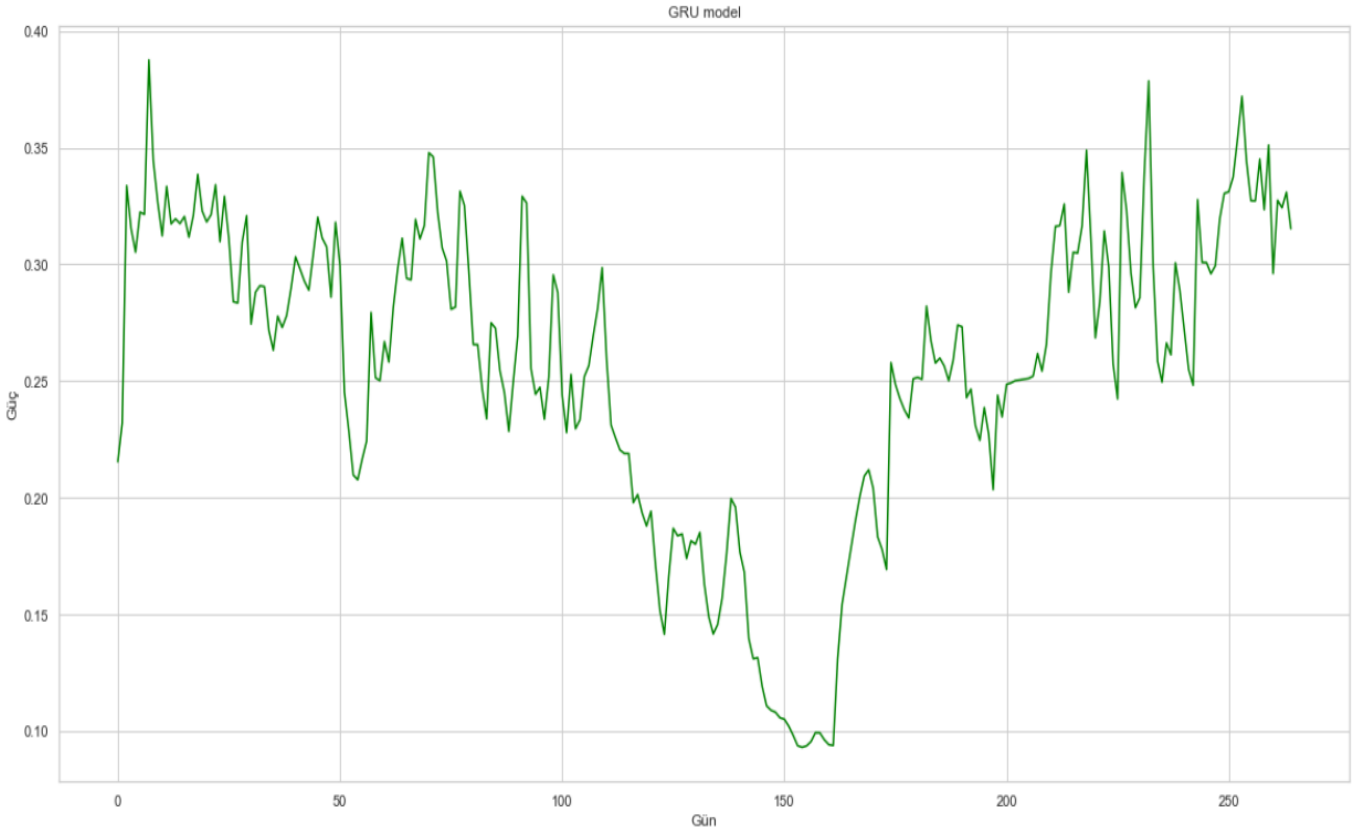
3.3.4 Tahmin Modeli ve Gerçek Veri ile Kıyaslanması

Tahmin Modeli

```
In [25]:
1 plt.figure(figsize=(20,10))
2 plt.plot(predicted, color="green")
3 plt.xlabel('Gün')
4 plt.ylabel('Güç')
5 plt.title("GRU model")
6 plt.show()
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plot içerisine 1 önceki bölümde eşitlediğimiz predicted(tahmin)'i giriyoruz ve color ile rengini green(yeşil) yapıyoruz.
- X ekseni üzerinde bir etiket oluşturuyoruz ve adını 'Gün' koyuyoruz.
- Y ekseni üzerinde bir etiket oluşturuyoruz ve adını 'Güç' koyuyoruz.
- Title ile grafiğe başlık olarak GRU model yazıyoruz.
- Plt.show ile ekrana gösteriyoruz.

Çıktısı;



Tahmin Modeli vs Gerçek Veri

```
In [26]:
1  plt.figure(figsize=(20,10))
2  plt.plot(true, color="red")
3  plt.plot(predicted, color="green")
4  plt.xlabel('Gün')
5  plt.ylabel('Güç')
6  plt.title("GRU model vs Gerçek Veri")
7  plt.show()
```

- Öncelikle figsize ile grafiğin en boy oranını belirledik.
- Plot içerisine true(Gerçek)'i giriyoruz ve color ile rengini red(kırmızı) yapıyoruz.
- Plot içerisine predicted(tahmin)'i giriyoruz ve color ile rengini green(yeşil) yapıyoruz.
- X ekseni üzerinde bir etiket oluşturuyoruz ve adını 'Gün' koyuyoruz.
- Y ekseni üzerinde bir etiket oluşturuyoruz ve adını 'Güç' koyuyoruz.
- Title ile grafiğe başlık olarak 'GRU model vs Gerçek Veri' yazıyoruz.
- Plt.show ile ekrana gösteriyoruz.

4 Sonuç

2011-2013 yılları arasında bir bölgenin kullandığı elektrik tüketimini gösteren bir veri seti ve Multivariable(Çok değişkenli) girişli 2006-2011 arasında bir bölgenin kullandığı elektrik tüketimini 'Jupyter Notebook' ile verileri ekrana göstererek daha sonra bu veri setleri ile veri görselleştirmesi, güzelleştirmesi ve düzenlemesi yapılmıştır. ARIMA ve LSTM modellerinde tek girişli veri seti, GRU Modelinde ise Multivariable(Çok değişkenli) veri seti kullanılmıştır. ARIMA ve LSTM aynı model içerisinde GRU ise ayrı model olarak yazıldı. Öncelikle her iki modeldede önce kütüphaneler belirlendi daha sonra veri seti ekrana yansıtılıp veri setini işledikten(düzenledikten) sonra grafikler ile güzelleştirme yapılıp modeller oluşturuldu. Son olarakta ARIMA ve LSTM'nin tahmin modelleri grafiklere dökülüp gerçek veri ile kıyaslatıldı. GRU modelinde ise gerçek değer ile GRU tahmin modeli kıyaslatıldı. Grafik üzerinde gösterme yöntemini kullandığımız fonksiyonlar Çizgi grafiği için 'lineplot', Kutu grafiği için 'boxplot', Histogram grafiği için 'distplot', Violin(Keman) Grafiği için 'violinplot' fonksiyonlarını kullanarak veri seti grafiklere döküldü. Daha sonra veri ön işleme adımları uygulanarak 3 model geliştirildi bu modeller tek değişkenli ARIMA ve LSTM, çok değişkenli GRU modeli. ARIMA ve LSTM modelleri yapıldıktan sonra önce birbirileri ile daha sonra gerçek veri ile 3 veri karşılaştırıldı. ARIMA modelinde otokorelasyon ve kısmi otokorelasyon grafikleri gösterildi. Gru modelinde Çeşitli grafiklerle veri görselleştirilmesi yapıldı. Tüm modellerde MSE ve RMSE değerleri hesaplandı. Karşılaştırma grafikleri için matplotlib kütüphanesi kullanıldı. Çeşitli metodlarla verileri hakkında bilgi edinildi. Sonuç olarak tüm modellerin tahmin grafikleri oluşturuldu ve LSTM ve ARIMA için bir biri ile karşılaştırıldı. Her modelde düzenlenen gerçek veriler o model ile karşılaştırıldı. Her modelin tahmin grafiği tek başına grafikte gösterildi. Bu yöntemle veri setleri üzerinde çeşitli tahmin modelleri oluşturulup sonuçları izlendi.

KAYNAKÇA

- https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/#:~:text=The%20statsmodels%20library%20provides%20an,model%20is%20additive%20or%20multiplicative.&text=The%20seasonal_decompose()%20function%20returns%20a%20result%20object.
- https://stackoverflow.com/questions/61040284/problems-with-acf-plots-and-operands
- https://numpy.org/doc/stable/reference/generated/numpy.diff.html
- https://www.researchgate.net/post/What-is-the-window-size-in-neural-networks-and-how-it-effects-training
- https://www.tutorialspoint.com/numpy/numpy_concatenate.htm#:~:text=numpy.-,concatenate,shape%20along%20a%20specified%20axis.
- https://tr.wikipedia.org/wiki/Dickey_Fuller_testi
- https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/
- https://dergipark.org.tr/tr/download/article-file/29738
- https://alkaline-ml.com/pmdarima/tips_and_tricks.html
- https://otexts.com/fpp2/seasonal-arima.html
- https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima_model.ARIMAResults.summary.html
- https://medium.com/@tuncerergin/keras-ile-derin-ogrenme-modeli-olusturma-4b4ffdc35323
- https://numpy.org/doc/stable/reference/generated/numpy.mean.html
- https://en.wikipedia.org/wiki/Additive_model
- https://en.wikipedia.org/wiki/Decomposition_of_time_series
- http://omz-software.com/pythonista/matplotlib/users/tight_layout_guide.html#:~:text=tight_layout%20automatically%20adjusts%20subplot%20params,%2C%20axis%20labels%2C%20and%20titles.
- https://www.statsmodels.org/stable/generated/statsmodels.graphics.tsaplots.plot_acf.html
- https://www.geeksforgeeks.org/python-coefficient-of-determination-r2-score/
- https://stackoverflow.com/questions/42763928/how-to-use-model-reset-states-in-keras#:~:text=If%20you%20use%20explicitly%20either,model%2C%20or%20layer.reset_states()
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

- https://towardsdatascience.com/pandas-for-people-in-a-hurry-59d966630ae0#:~:text=Exploring%20DataFrame,5%20rows%20of%20the%20dataframe.&text=tail()%20Returns%20the%20last%205%20rows%20of%20the%20dataframe.
- https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/
- https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/
- https://www.analyticsvidhya.com/blog/2020/10/how-to-create-an-arima-model-for-time-series-forecasting-in-python/
- https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/#:~:text=ARIMA%2C%20short%20for%20'AutoRegressive%20Integrated,to%20predict%20the%20future%20values.
- https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://ishakdolek.medium.com/lstm-d2c281b92aac#:~:text=LSTM(Long%2D%20Short%20Term%20Memory,taraf%C4%B1ndan%20at%C4%B1f%20ald%C4%B1%20ve%20yayg%C4%B1nla%C5%9Ft%C4%B1r%C4%B1ld%C4%B1.
- https://www.rdocumentation.org/packages/statisticalModeling/versions/0.3.0/topics/evaluate_model
- https://www.researchgate.net/post/Can-anyone-explain-batch-size-batch-input-shape-return-sequenceTrue-False-in-python-during-training-LSTM-with-KERAS#:~:text=batch_size%20denotes%20the%20subset%20size,appliance%20of%20the%20previous%20batch.
- https://stackoverflow.com/questions/54009661/what-is-the-timestep-in-keras-lstm
- https://blog.floydhub.com/gru-with-pytorch/
- https://en.wikipedia.org/wiki/Gated_recurrent_unit
- https://paperswithcode.com/method/gru
- https://www.geeksforgeeks.org/python-pandas-to_datetime/
- https://www.geeksforgeeks.org/matplotlib-axis-axis-get_ticklabels-function-in-python/
- https://stackoverflow.com/questions/61443261/what-is-the-use-of-pd-plotting-register-matplotlib-converters-in-pandas
- https://nextjournal.com/blog/plotting-pandas-prophet
- https://pythonbasics.org/seaborn-distplot/#:~:text=We%20use%20seaborn%20in%20combination,()%20and%20rugplot()%20functions.

- https://www.geeksforgeeks.org/matplotlib-figure-figure-add_subplot-in-python/#:~:text=matplotlib.-,figure.,part%20of%20a%20subplot%20arrangement.
- https://literarydevices.net/subplot/
- https://pythonprogramming.net/subplot2grid-add_subplot-matplotlib-tutorial/
- https://www.splashlearn.com/math-vocabulary/geometry/line-plot#:~:text=A%20Line%20plot%20can%20be,the%20frequency%20of%20each%20value.
- https://tr.wikipedia.org/wiki/Kutu_grafi%C4%9Fi
- https://www.mathsisfun.com/data/histograms.html#:~:text=Histogram%3A%20a%20graphical%20display%20of,many%20fall%20into%20each%20range.
- https://en.wikipedia.org/wiki/Violin_plot