

DERİN ÖĞRENME YÖNTEMLERİ İLE ELEKTRİK TÜKETİMİNİN TAHMİNİ

DENİZ CAN TOŞUR

ANTALYA,2021

ÖZET

Bu rapor çalışmasında, 2004-2018 yılları arasında bir bölgenin kullandığı elektrik tüketimini gösteren bir veri seti kullanarak 'Jupyter Notebook' ile verileri ekrana gösterip daha sonra bu veri seti ile veri görselleştirmesi, güzelleştirmesi ve düzenlemesi yapılarak verileri grafikler ile görselleştirme işlemi yapılmıştır. İlk olarak veri setini 'Jupyter Notebook' ile kod yazma kısmına kütüphaneler belirlenip yazılıp daha sonra veri setinin ilk beş kısmı ile son beş kısmı ekrana yazılıp daha sonra veri seti ile ilgili önemli bilgileri ekrana yazdırdığımız kod bloklarını anlatacaktır. Bir sonraki işlemde bu veri setini 'Ay', 'Yıl', 'Tarih', 'Saat', 'Hafta', 'Gün' şeklinde veri güzelleştirmesi yapılarak veri hakkında daha çok bilgi edinilmiştir ve daha güzel bir görüntü sağlanmıştır. Veri kümemizde kaç tane benzersiz yıl olduğunu bulmak için 'unique()' kod bloğumuzu kullanıp toplam benzersiz yıllar bulunmuş oldu. Kütüphanelerimizin yardımıyla gerekli grafik kodları yazılıp veri seti grafik üzerinde gösterilmiştir. Bu 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. Veri setini günlere göre ayarlanıp oluşan veri den son 100 veri çıkartılıp Yeni oluşan veri seti LSTM katmanları oluşturularak Epoch'unda yardımıyla makine eğitildi ve yeni bir tahmini 100 günlük model oluşturuldu. Daha sonra çıkardığımız veri setinden çıkardığımız son 100 veri ile tahmin modelinin grafiği tek tek ekrana yansıtıldı. Bu iki grafiği daha sonra aynı grafiğe koyarak karşılaştırma yapıldı.

Anahtar Kelimeler: unique(), lineplot, boxplot, distplot, violinplot, LSTM, Epoch, karşılaştırma.

ABSTRACT

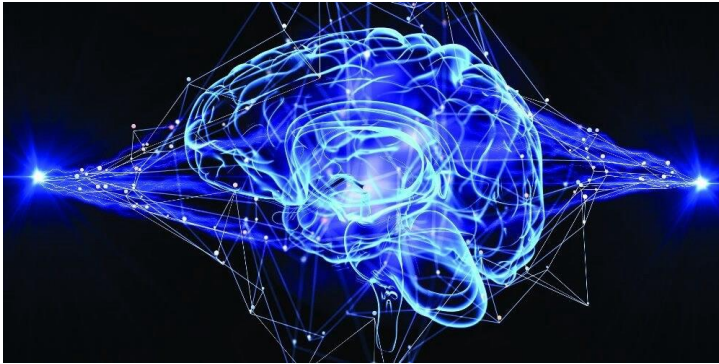
In this study, data visualization, beautification and editing were done with 'Jupyter Notebook' using a data set between 2004-2018, and visualization process was performed with its feeds. First, the lectures will be explained to the writing part of the data set with 'Jupyter Notebook', then the first five parts and the last five parts of the data set will be written, and then the important information about the data set will be explained important information about the software, important information about the software. In the next process, more information about the data was obtained and a better view was provided by making data beautify such as "Month", "Year", "Date", "Hour", "Week", "Day". Using our 'unique ()' code block to find out how many unique years are in our dataset, the total unique years have been found. Graphical analysis of the data set that our libraries can protect. Using the methods of displaying on this graph, the data set was poured into the graphs by using the functions 'lineplot' for line graph, 'boxplot' for box graph,

'distplot' for Histogram graph, 'violinplot' for Violin Graph. By arranging the data set according to days, extracting the last 100 data from the data, creating the newly created data set LSTM layers, the construction machine was trained in Epoch and a new estimated 100-day model was created. The graph of the prediction model was projected on a single screen with the last 100 data we extracted from the data set we extracted later. These two graphs were then compared to the same graph.

Keywords: unique(), lineplot, boxplot, distplot, violinplot, LSTM, Epoch, comparison.

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 dağıtımını 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 modeli 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 bir tahmin yapabilmemiz için öncelikle bir veri setine ihtiyacımız bulunmaktadır. Kullanılan veri setinde “2004” yılı ile “2018” tarihleri arasında tüketilen saatlik enerji tüketimini gösteren bir veri seti ele alınmıştır. Veri seti tam “121275” veriden oluşmakta ve bu veri setimiz bir “txt” dosya uzantısı şeklindedir. 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.

 MW_hourly - Not Defteri

[Dosya](#) [Düzen](#) [Biçim](#) [Görünüm](#) [Yardım](#)

Datetime,MW

2004-12-31 01:00:00,1596.0

2004-12-31 02:00:00,1517.0

2004-12-31 03:00:00,1486.0

2004-12-31 04:00:00,1469.0

2004-12-31 05:00:00,1472.0

Yukarıdaki veriler veri setinin ilk 5 verisini içermektedir. Veri setinde Zaman(Datetime) kısmında bir bölüm ve Mega Watt(MW) kısmında ikinci bölüm oluşturuldu. Arasındaki virgül iki ayrı bölüm olduklarını ayırmak için kullanıldı. Bu kısımlar çok önemlidir çünkü kodlama yaparken verileri tek tek yazmak yerine sadece bölüm isimlerini yazmak bize zamandan kazandıracaktır. Bölümlerin altındaki veriler tarih,ay,gün,saat şeklinde yazılıp birinci bölümü oluşturuldu. Virgülden sonraki kısmında tüketilen elektrik gücünü yani MW’ı temsil etmektedir. Buradaki verilerin altında bu veriler hariç tam 121270 tane daha veri bulunmaktadır.

2.2 Gerekli Kütüphaneler

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

Pandas, Python programlama dili için kullanımı kolay olmakla birlikte veri analizi araçlarını kullanılmasını sağlayan açık kaynaklı bir Python kütüphanesidir. Veri setindeki verileri analiz

etmek için pandas kütüphanesinden yardım alınmıştır. Pandas kütüphanesini çağırmak için kullanılan kod “import pandas as pd” yazılır. Pandas kütüphanesini kısaltmak için “as pd” ile bir kısaltma işlemi yapılır. Kısaltma işleminin faydası artık kodlamada “pd” yazılması yeterli olacaktır. Kısaltma işlemi kodları yazarken kolaylık sağlamıştır.

Numpy, hesaplamaları hızlı bir şekilde yapılmasını sağlayan bir matematik kütüphanesidir. Numpy kütüphanesi çağırmak için kullanılan kod “import numpy as np” yazılır. Numpy kütüphanesine kısaltma işlemi yazmak için “as np” yazılır. Yine aynı şekilde bu kısaltma işlemi kodlama yaparken zamandan kazanmak için yazılmıştır ve bu sayede kolaylık sağlanmıştır.

Matplotlib.pyplot, python programlama dili için durum tabanlı bir arayüzdür. MATLAB gibi bir grafik çizilmesine olanak sağlar. Matplotlib.pyplot kütüphanesini çağırmak için kullanılan kod “import matplotlib.pyplot as plt” yazılır ve pyplot kütüphanesi çağırılır. Matplotlib.pyplot kütüphanesini kısaltmak için “as plt” yazılır ve kısaltma işlemi yapılır. Veri setindeki verileri grafiğe dökmek için pyplot olmazsa olmazdır.

Seaborn, matplotlib tabanlı bir Python veri görselleştirme kütüphanesidir. Python için popüler Matplotlib veri görselleştirme kitaplığının üzerinde çalışan bir veri görselleştirme kitaplığıdır. Matplotlib çizimlerini daha güzel gösterir. Seaborn kütüphanesini çağırmak için kullanılan kod “import Seaborn as sns” yazılır. Seaborn kütüphanesini kısaltmak için “as sns” yazılır ve kısaltma işlemi yapılır.

Pprint, veri yapılarınızın estetik açıdan hoş temsillerini üretmek için "güzel bir yazıcı" içerir. Biçimlendirici, yorumlayıcı tarafından doğru bir şekilde ayrıştırılabilen veri yapılarının temsillerini üretir ve ayrıca bir insan tarafından okunması kolaydır. Çıktı, mümkünse tek bir satırda tutulur ve birden çok satıra bölündüğünde girintilenir.

%Matplotlib inline, kod bloğu sadece Matplotlib’in etkileşimli modunu kurar.

3. MATERYAL VE YÖNTEM

Öncelikle veri setini ekrana yazdırma işlemi gerçekleştirildi.

```
In [2]: df = pd.read_csv("MW_hourly.txt")
print("="*50)
print("İlk Bes Sıra ", "\n")
print(df.head(5), "\n")
print("="*50)
print("Son Bes Sıra ", "\n")
print(df.tail(5), "\n")
print("="*50)
print("Veri Kümesi Hakkında Bilgi", "\n")
print(df.info(), "\n")

print("="*50)
print("Veri Kümesini Tanımlayalım (Kategorik tür) ", "\n")
print(df.describe(), "\n")

print("="*50)
print("Bos Değerler t ", "\n")
print(df.isnull().sum(), "\n")
```

- 1- Öncelikle 'pd.read_csv' işlemi bir CSV dosyasını yada Txt dosyanı veri çerçevesi ortamına aktarır.
- 2- Veriler hakkında bilgileri ayırmak için aralara '=' ifadeden 50 tane koyulmuştur.
- 3- İlk veriler için bir başlık yazılmıştır(print("İlk Bes Sira", "\n")).
- 4- df.head(5) işlemi ile sütun isimlerini içeren satır numarası yazılmış oldu.
- 5- Veriler hakkında bilgileri ayırmak için aralara '=' ifadeden 50 tane koyulmuştur.
- 6- Diğer veriler için bir başlık yazılmıştır(print("Son Bes Sira", "\n")).
- 7- df.tail(5) işleminde tail metodu son 5 satırı döndürdüğü için kullanıldı.
- 8- İlk veriler için bir başlık yazılmıştır(print("İlk Bes Sira", "\n")).
- 9- Diğer veriler için bir başlık yazılmıştır(print("Veri kümesi hakkında bilgi", "\n")).
- 10- df.info metodu ile veri seti hakkında bilgi edinmemizi sağlar. df.info her bir sütunda null olmayan girdi sayısı ve sütunun veri tipi gibi bilgileri elde edebiliriz.
- 11- Veriler hakkında bilgileri ayırmak için aralara '=' ifadeden 50 tane koyulmuştur.
- 12- Diğer veriler için bir başlık yazılmıştır(print("Veri kümesini tanımlayalım(Kategorik tür)", "\n")).
- 13- df.describe() metodu sayısal verilere sahip olan sütunların max, min , std...gibi istatistiksel değerlerini döndürür. Bu yüzden veri setinde daha çok bilgi edinilmek için kullanıldı.
- 14- Veriler hakkında bilgileri ayırmak için aralara '=' ifadeden 50 tane koyulmuştur.
- 15- Diğer veriler için bir başlık yazılmıştır(print("Bos Değerler t", "\n")).
- 16- df.isnull() veri setinde boş değer olup olmadığını kontrol etmemizi sağlıyor.

Çıktısı;

```
=====
ilk Bes Sıra

   Datetime    MW
0  2004-12-31 01:00:00  1596.0
1  2004-12-31 02:00:00  1517.0
2  2004-12-31 03:00:00  1486.0
3  2004-12-31 04:00:00  1469.0
4  2004-12-31 05:00:00  1472.0

=====
Son Bes Sıra

   Datetime    MW
121270 2018-01-01 20:00:00  2732.0
121271 2018-01-01 21:00:00  2724.0
121272 2018-01-01 22:00:00  2664.0
121273 2018-01-01 23:00:00  2614.0
121274 2018-01-02 00:00:00  2552.0

=====
Veri Kümesi Hakkında Bilgi

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 121275 entries, 0 to 121274
Data columns (total 2 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   Datetime    121275 non-null   object
1   MW          121275 non-null   float64
dtypes: float64(1), object(1)
memory usage: 1.9+ MB
None

=====
Veri Kümesini Tanımlayalım (Kategorik tür)

      MW
count 121275.000000
mean  2037.851140
std    393.403153
min    982.000000
25%   1749.000000
50%   2009.000000
75%   2279.000000
max   3746.000000

=====
Bos Değerler t

Datetime    0
MW          0
dtype: int64
```

3.1 VERİ SETİNDE TARİH VE SAAT SÜTUNLARINI YENİDEN BİÇİMLENDİRME

Veri seti zamana bağlı bir veri olduğu için zamana göre düzenlemek veri görselleştirmesi veya güzelleştirmesi için daha yararlı olacaktır.

```
In [3]: # Yılın Ayı, Günü, Saati vb. Gibi Tüm Verileri Çıkartıyoruz
dataset = df
dataset["Ay"] = pd.to_datetime(df["Datetime"]).dt.month
dataset["Yıl"] = pd.to_datetime(df["Datetime"]).dt.year
dataset["Tarih"] = pd.to_datetime(df["Datetime"]).dt.date
dataset["Saat"] = pd.to_datetime(df["Datetime"]).dt.time
dataset["Hafta"] = pd.to_datetime(df["Datetime"]).dt.week
dataset["Gün"] = pd.to_datetime(df["Datetime"]).dt.day_name()
dataset = df.set_index("Datetime")
dataset.index = pd.to_datetime(dataset.index)
dataset.head(1)
```

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

- 3- dataset içerisinde başlığımızı yazıp(Yıl) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'year' yani yıl verilerini buraya yazmasını istedik.
- 4- dataset içerisinde başlığımızı yazıp(Tarih) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'date' yani tarih verilerini buraya yazmasını istedik.
- 5- dataset içerisinde başlığımızı yazıp(Saat) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'time' yani saat verilerini buraya yazmasını istedik.
- 6- dataset içerisinde başlığımızı yazıp(Hafta) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'week' yani hafta verilerini buraya yazmasını istedik.
- 7- dataset içerisinde başlığımızı yazıp(Gün) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'day_name()' yani gün verilerini buraya yazmasını istedik.
- 8- set_index() işlevi, var olan sütunları kullanarak veri çerçevesi dizinini ayarlamak için kullanılır.("Datetime") ile bu veri çerçevesine bir satır etiketi(başlık) olarak ayarlandı.
- 9- to_datetime, dize tarih saatini python tarih saat nesnesine dönüştürülmesini sağladı.
- 10- dataset.head(1) ile sadece işlemlerin hepsini yazdırmak yerine sadece ilk satırını ekrana yazdırdık.

Çıktısı;

	MW	Ay	Yıl	Tarih	Saat	Hafta	Gün
Datetime							
2004-12-31 01:00:00	1596.0	12	2004	2004-12-31	01:00:00	53	Friday

3.2 VERİ SETİNDE KAÇ TANE BENZERSİZ AY VE YIL OLDUĞUNUN BULUNMASI

Veri setinde kaç tane ay ve yıl olduğunu öğrenmek, verimizi daha iyi yorumlamamıza yardımcı olacaktır.

```
In [4]: # Veri Kümesinde Kaç Benzersiz Yılımız Var
print(df.Yıl.unique(), "\n")
print("Toplam Benzersiz Yıl Sayisi", df.Yıl.nunique(), "\n")

[2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
 2018]

Toplam Benzersiz Yıl Sayisi 15
```

- 1- Öncelikle df veri setimizin Yıl bölümünde unique() işlemi yapacağız. Unique(), birbirinden farklı kaç kategori olduğunun bilgisini bize sunuyor, o yüzden bu işlemi Yıl kategorisi üzerinde yapıp kaç farklı yıl olduğunu bulduk ve ekrana yazdırdık.
- 2- 'Toplam Benzersiz Yıl sayısı' adında ekrana yazı yazdırıp benzersiz yılların toplamını sonuca yazdırdık.


```
In [5]: # Veri Kümesinde Kaç Benzersiz Yılımız Var
print(df.Ay.unique(),"\n")
print("Toplam Benzersiz Ay Sayisi:", df.Ay.nunique(), "\n")
print("Toplam Ay Sayisi:", df.Ay.nunique()*df.Yil.nunique(), "\n")
```

```
[12  1 11 10  9  8  7  6  5  4  3  2]
```

```
Toplam Benzersiz Ay Sayisi: 12
```

```
Toplam Ay Sayisi: 180
```

- 1- Öncelikle df veri setimizin Ay bölümünde unique() işlemi yapacağız. Unique(), birbirinden farklı kaç kategori olduğunun bilgisini bize sunuyor, o yüzden bu işlemi Ay kategorisi üzerinde yapıp kaç farklı Ay olduğunu bulduk ve ekrana yazdırdık.
- 2- ‘Toplam Benzersiz Ay Sayisi’ adında ekrana yazı yazdırıp kaç tane benzersiz ay olduğunu ekrana yazdırdık.
- 3- ‘Toplam Ay sayısı’ adında ekrana yazı yazdırıp basit bir matematiksel işlem ile benzersiz Ay sayısı ile benzersiz yıl sayısını çarpıp ekrana yazdırdık

3.3 VERİ KÜMESİNİ GRAFİKLER İLE GÖSTERİMİ

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 sınırlandıran belirli özelliklere sahip olabilir. Bu bölümde veri setimizi grafiklere dökmeyi göstereceğiz.

3.3.1 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 [6]: #Çizgi Grafiği
from matplotlib import style

fig = plt.figure()
ax1 = plt.subplot2grid((1,1), (0,0))

style.use('ggplot')

sns.lineplot(x=dataset["Yil"], y=dataset["MW"], data=df)
sns.set(rc={'figure.figsize':(25,6)})

plt.xlabel("Tarih")
plt.ylabel("Mega Watt cinsinden enerji")
plt.grid(True)
plt.legend()

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

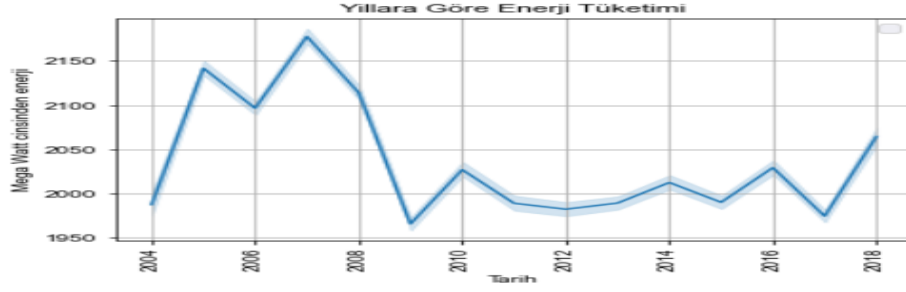
plt.title("Yıllara Göre Enerji Tüketimi")

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

- 1- Öncelikle stlye kütüphanesini çağırdık(import ettik).
- 2- plt.figure() ile 'fig' adında bir figür oluşturduk.
- 3- plt.subplot2grid bir eksen nesnesi oluşturmak için bize esneklik sağlayacaktır.
- 4- R dili görselleştirmesi olan ggplot'u style.use('ggplot') diyerek ggplot stilini kullandık.
- 5- sns.lineplot ile seaborn kütüphanesinden çizgi grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setimizden 'Yil' kısmını yazdırdık, y değerine de veri setimizden 'MW' kısmını kullanıp yazdırdık.
- 6- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- 7- Grafiğin x koordinatı altına etiket olarak 'Tarih' yazdırdık.
- 8- Grafiği y koordinatı altınada etiket olarak 'Mega Watt cinsinden enerji' yazdırdık.
- 9- Grafikte ızgara çizgilerinin gözükmesini istediğimiz için 'plt.grid(True)' kısmının içini doğru(true) yaptık.
- 10- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- 11- Daha sonra grafiğe kene etiketi('ax1.xaxis.get_ticklabels()') ekleyip 'Yıllara Göre Enerji Tüketimi' adında başlık yazdırıyoruz.

Çıktısı;

```
Text(0.5, 1.0, 'Yıllara Göre Enerji Tüketimi')
```



3.3.2 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
from matplotlib import style

fig = plt.figure()
ax1 = plt.subplot2grid((1,1), (0,0))

style.use('ggplot')

sns.boxplot(x=dataset["Yıl"], y=dataset["MW"], data=df)
sns.set(rc={'figure.figsize':(15,6)})

plt.title("2004 Yılında Enerji Tüketimi")
plt.xlabel("Tarih")
plt.ylabel("Mega Watt cinsinden enerji")
plt.grid(True)
plt.legend()

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

plt.title("Yıllara Göre Enerji Tüketimi")
```

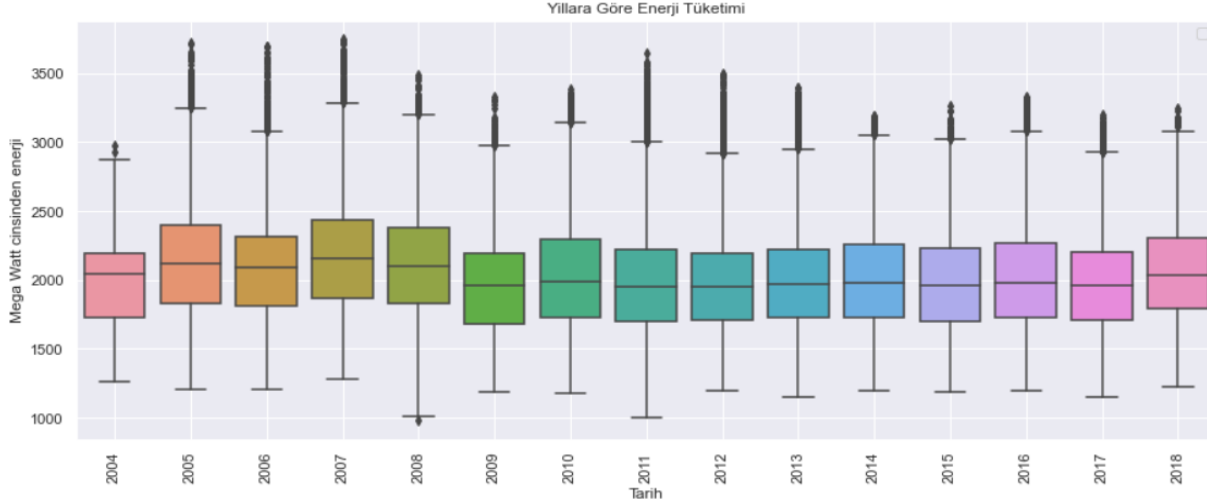
No handles with labels found to put in legend.

- 1- Öncelikle stlye kütüphanesini çağırdık(import ettik).
- 2- plt.figure() ile 'fig' adında bir figür oluşturduk.
- 3- plt.subplot2grid bir eksen nesnesi oluşturmak için bize esneklik sağlayacaktır.
- 4- R dili görselleştirmesi olan ggplot'u style.use('ggplot') diyerek ggplot stilini kullandık.
- 5- sns.boxplot ile seaborn kütüphanesinden kutu grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setimizden 'Yıl' kısmını yazdırdık, y koordinatına da veri setimizden 'MW' kısmını kullanıp yazdırdık.
- 6- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- 7- Grafiğin x koordinatı altına etiket olarak 'Tarih' yazdırdık.
- 8- Grafiği y koordinatı altınada etiket olarak 'Mega Watt cinsinden enerji' yazdırdık.
- 9- Grafikte ızgara çizgilerinin gözükmesini istediğimiz için 'plt.grid(True)' kısmının içini doğru(true) yaptık.
- 10- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.

11- `axis.get_ticklabels()` ile `ax1` figürü üzerinde eksen modülünün listelerine kene etiketi almak için kullandık. Daha sonra ‘Yıllara Göre Enerji Tüketimi’ adında başlık yazdırıyoruz.

Çıktısı;

`Text(0.5, 1.0, 'Yıllara Göre Enerji Tüketimi')`



3.3.3 Üçlü Çizgi Grafiği

Üçlü çizgi grafiği, çizgi grafiğinin boyutları ayrılp düzenlenmiş şekilde 3 farklı olanıdır.

```
In [7]: from matplotlib import style

fig = plt.figure()

plt.title("Enerji tüketimi")
ax1= fig.add_subplot(311)
ax2= fig.add_subplot(312)
ax3= fig.add_subplot(313)

style.use('ggplot')

y_2004 = dataset["2004"]["MW"].to_list()
x_2004 = dataset["2004"]["Tarih"].to_list()
ax1.plot(x_2004,y_2004, color="green", linewidth=2.4)

y_2005 = dataset["2005"]["MW"].to_list()
x_2005 = dataset["2005"]["Tarih"].to_list()
ax2.plot(x_2005, y_2005, color="green", linewidth=1)

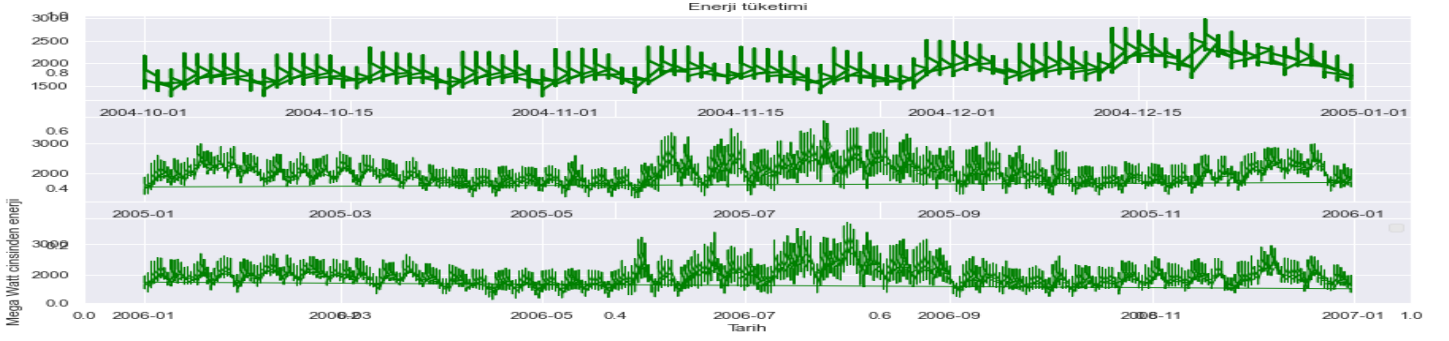
y_2006 = dataset["2006"]["MW"].to_list()
x_2006 = dataset["2006"]["Tarih"].to_list()
ax3.plot(x_2006, y_2006, color="green", linewidth=1)

plt.rcParams["figure.figsize"] = (18,8)
plt.xlabel("Tarih")
plt.ylabel("Mega Watt cinsinden enerji")
plt.grid(True, alpha=1)
plt.legend()
```

- 1- Öncelikle `stlye` kütüphanesini çağırdık(import ettik).
- 2- `plt.figure()` ile ‘fig’ adında bir figür oluşturduk.

- 3- Bu figürümüze 'title' metodu ile başlık ekledik
- 4- fig.add_subplot() komutu ile ax1,ax2,ax3 adında 3 tane eksen eklendi.
- 5- R dili görselleştirmesi olan ggplot'u style.use('ggplot') diyerek ggplot stilini kullandık.
- 6- y_2004 ile veri setindeki 2004 yılının mega watt kısmını(MW) belirleyip listeye ekliyoruz. Bu kısım grafikte y koordinatını temsil edecektir.
- 7- x_2004 ile veri setindeki 2004 yılı tarihli olan tüm zaman serilerini(Tarih) belirleyip listeye ekliyoruz. Bu kısım grafikte x koordinatını temsil edecektir.
- 8- ax1.plot ile grafiğin yukarıda belirttiğimiz x,y kısmını yazıp linewidth ile genişliği belirleyerek oluşturuyoruz.
- 9- 2.grafik için y_2005 ile veri setindeki 2005 yılının mega watt kısmını(MW) belirleyip listeye ekliyoruz. Bu kısım grafikte y koordinatını temsil edecektir.
- 10- 2. grafik için x_2005 ile veri setindeki 2005 yılı tarihli olan tüm zaman serilerini(Tarih) belirleyip listeye ekliyoruz. Bu kısım grafikte x koordinatını temsil edecektir.
- 11- ax2.plot ile grafiğin yukarıda belirttiğimiz x,y kısmını yazıp linewidth ile genişliği belirleyerek oluşturuyoruz.
- 12- 3.grafik için y_2005 ile veri setindeki 2005 yılının mega watt kısmını(MW) belirleyip listeye ekliyoruz. Bu kısım grafikte y koordinatını temsil edecektir.
- 13- 3. grafik için x_2005 ile veri setindeki 2005 yılı tarihli olan tüm zaman serilerini(Tarih) belirleyip listeye ekliyoruz. Bu kısım grafikte x koordinatını temsil edecektir.
- 14- ax3.plot ile grafiğin yukarıda belirttiğimiz x,y kısmını yazıp linewidth ile genişliği belirleyerek oluşturuyoruz.
- 15- Figürü yapılandırmak için plt.rcParams kullandık ve en,boy oranını belirledik.
- 16- xlabel ile x koordinatı altına 'Tarih' etiketi yazdırıyoruz.
- 17- ylabel ile y koordinatı altına 'Mega Watt cinsinden enerji' etiketi yazdırıyoruz.
- 18- Grafikte ızgara çizgilerinin gözükmemesini istediğimiz için 'plt.grid(True)' kısmının içini doğru(true) yapıp çizgi türünü belirlemek için ise 'alpha=1' dedik.
- 19- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- 20- Not: Aynı yöntem ile 2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018 yıllarında yapılabilir.

Çıktısı;



3.3.4 Histogram Grafiği

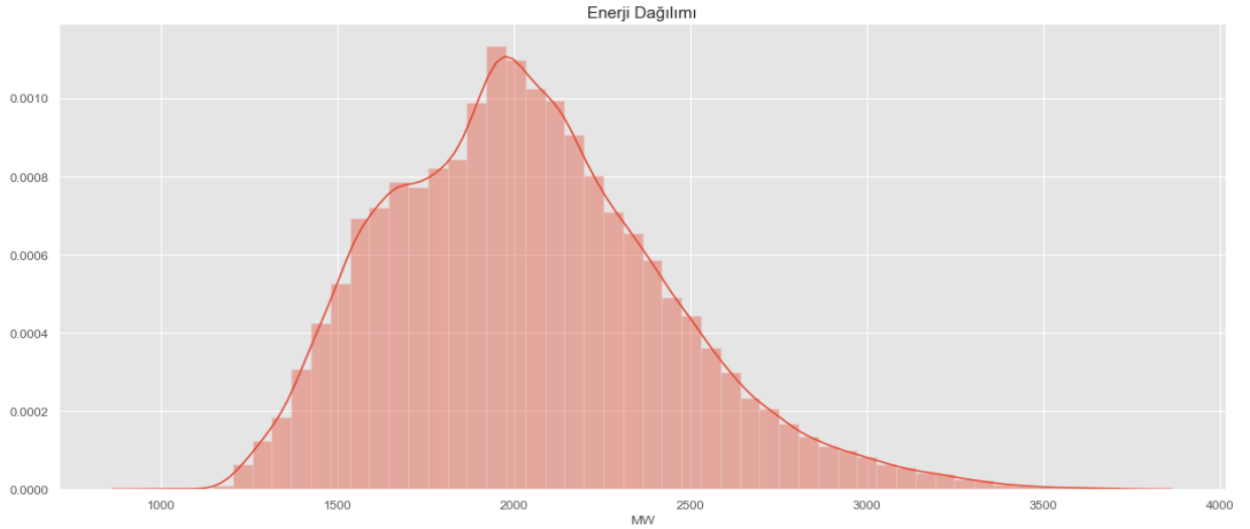
Çubuk Grafiğine benzer , ancak histogram sayıları aralıklar halinde gruplandırır . Her çubuğun yüksekliği, her bir aralığa kaç tane düştüğünü gösterir. Ve hangi aralıkları kullanacağımıza biz karar veririz.

```
In [8]: #Histogram Grafiği
sns.distplot(dataset["MW"])
plt.title("Enerji Dağılımı")
```

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

Çıktısı;

```
Out[8]: Text(0.5, 1.0, 'Enerji Dağılımı')
```



3.3.5 Zamana Göre Enerji Grafiği

Zamana göre enerji grafiği, Çizgi grafiğini kullanılarak yapılmıştır. Daha önceki örnekte ‘Yıl’ yazılırken burda ‘Saat’ yazılır.

```
In [9]: pd.plotting.register_matplotlib_converters()
```

Öncelikle grafiklerlerimiz gelecekteki uyarı(FutureWarning) sorunu yaşamaması için `plotting.register_matplotlib_converters()` metodunu yazıyoruz.

```
In [10]: #Zamana Göre Çizgi Grafiği
fig = plt.figure()
ax1= fig.add_subplot(111)

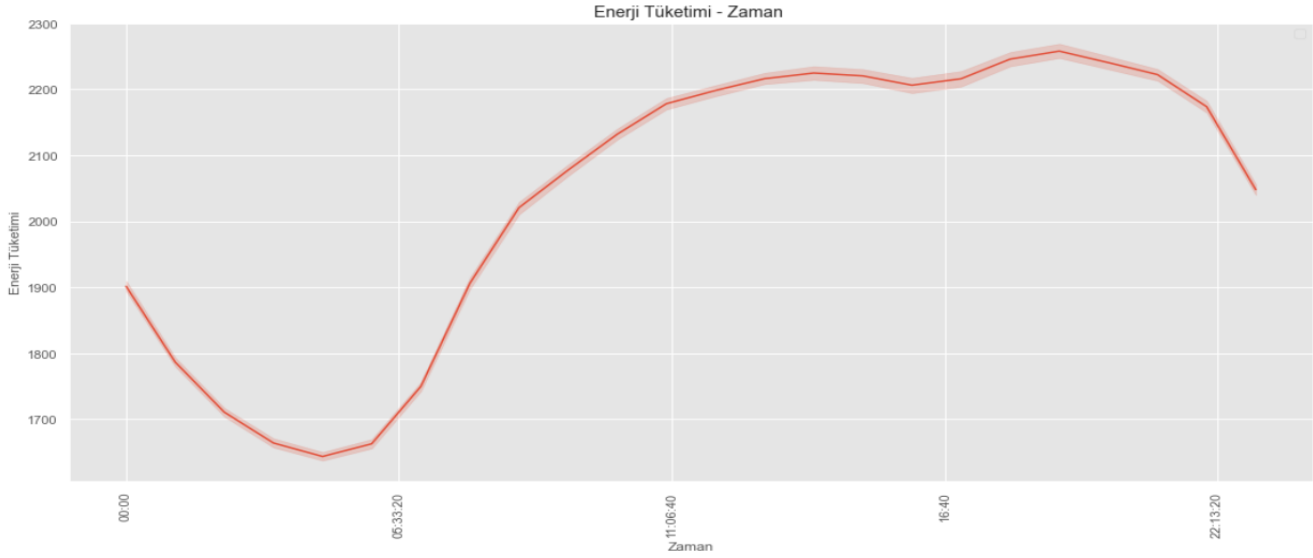
sns.lineplot(x=dataset["Saat"],y=dataset["MW"], data=df)
plt.title("Enerji Tüketimi - Zaman ")
plt.xlabel("Zaman")
plt.ylabel("Enerji Tüketimi")
plt.grid(True, alpha=1)
plt.legend()

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

No handles with labels found to put in legend.

- 1- `plt.figure()` ile ‘fig’ adında bir figür oluşturduk.
- 2- `fig.add_subplot()` komutu ile `ax1` eksen eklendi.
- 3- `sns.lineplot` ile seaborn kütüphanesinden çizgi grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setimizden ‘Saat’ kısmını yazdırdık, y koordinatına da veri setimizden ‘MW’ kısmını kullanıp yazdırdık.
- 4- `plt.title()` ile ‘Enerji Tüketimi – Zaman’ adında başlık oluşturduk.
- 5- `plt.xlabel()` ile x koordinatı altına ‘Zaman’ etiketi ekledik.
- 6- `plt.ylabel()` ile y koordinatı altına ‘Enerji Tüketimi’ etiketi ekledik.
- 7- Grafikte ızgara çizgilerinin gözükmesini istediğimiz için ‘`plt.grid(True)`’ kısmının içini doğru(true) yapıp çizgi türünü belirlemek için ise ‘`alpha=1`’ dedik.
- 8- `plt.legend()` diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- 9- `axis.get_ticklabels()` ile `ax1` figürü üzerinde eksen modülünün listelerine kene etiketi almak için kullandık.

Çıktısı;



3.3.6 Zamana Göre Kutu Grafiği

Zamana göre Kutu grafiği, Çizgi grafiğini kullanılarak yapılmıştır. Daha önceki örnekte ‘Yıl’ yazılırken burda ‘Saat’ yazılır.

```
In [16]: #Zamana Göre Kutu Grafiği
fig = plt.figure()
ax1= fig.add_subplot(111)

sns.boxplot(x=dataset["Saat"],y=dataset["MW"], data=df)
plt.title("Enerji Tüketimi - Zaman ")
plt.xlabel("Zaman")
plt.ylabel("MW")
plt.grid(True, alpha=1)
plt.legend()

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

No handles with labels found to put in legend.

- 1- plt.figure() ile ‘fig’ adında bir figür oluşturduk.
- 2- fig.add_subplot() komutu ile ax1 eksenini ekledik.
- 3- sns.boxplot ile seaborn kütüphanesinden kutu grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setimizden ‘Saat’ kısmını yazdırdık, y koordinatına da veri setimizden ‘MW’ kısmını kullanarak yazdırdık.
- 4- plt.title() ile ‘Enerji Tüketimi – Zaman’ adında başlık oluştur.
- 5- plt.xlabel() ile x koordinatı altına ‘Zaman’ etiketi ekledik.
- 6- plt.ylabel() ile y koordinatı altına ‘MW’ etiketi ekledik

- 7- Grafikte ızgara çizgilerinin gözükmesini istediğimiz için 'plt.grid(True)' kısmının içini doğru(true) yapıp çizgi türünü belirlemek için ise 'alpha=1' dedik.
- 8- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.
- 9- axis.get_ticklabels() ile ax1 figürü üzerinde eksen modülünün listelerine kene etiketi almak için kullandık.

3.3.7 Violin(Keman) Grafiği

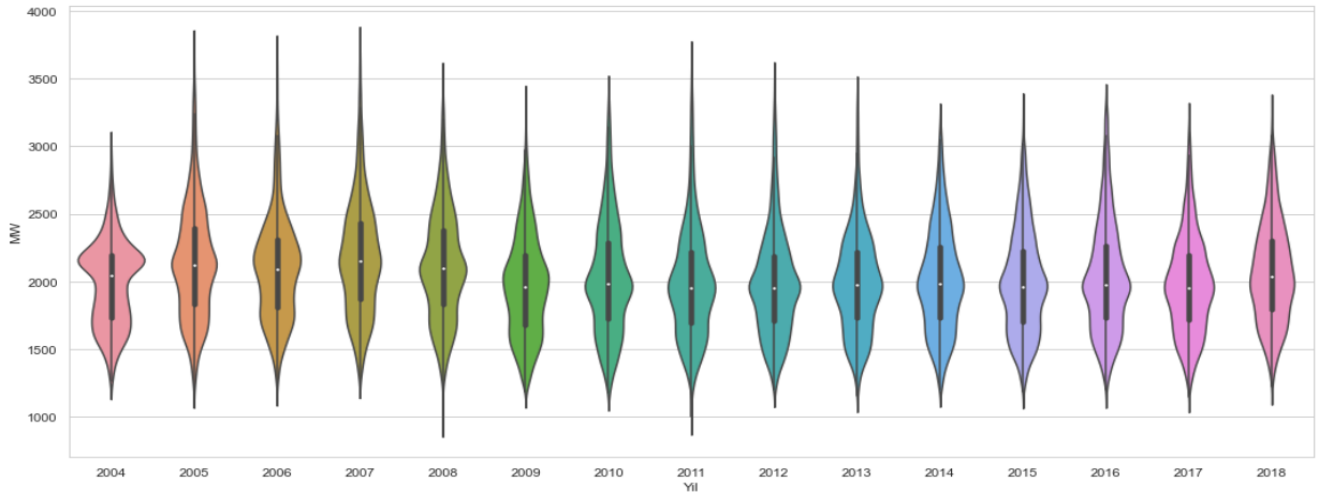
Bir keman grafiği , sayısal verileri çizme yöntemidir. Her bir tarafa döndürülmüş bir çekirdek yoğunluğu grafiğinin eklenmesiyle bir kutu grafiğine benzer . Keman grafikleri , genellikle bir çekirdek yoğunluğu tahmincisi tarafından düzleştirilen farklı değerlerde verilerin olasılık yoğunluğunu göstermeleri 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 [13]: #Violin(keman) Grafiği
sns.set_style('whitegrid')
sns.violinplot(x = 'Yil', y = 'MW', data = df)
```

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

Çıktısı;

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1781f967fa0>
```



Test ve Train İşlemleri

```
In [15]: NewDataSet = dataset.resample('D').mean()
```

- 1- Veri setini Günler olarak düzenleyip Toplam Kaç gün olduğunu resample metodu ile hesapladık ('D')

```
In [16]: print("Old Dataset ",dataset.shape ) # Veri setimizin eski veri sayısı  
print("New Dataset ",NewDataSet.shape ) # Veri setinin yeni veri sayısı (Gün bazında)
```

```
Old Dataset (121275, 7)  
New Dataset (5055, 4)
```

```
In [17]: TestData = NewDataSet.tail(100)  
Training_Set = NewDataSet.iloc[:,0:1]  
Training_Set = Training_Set[:-60]
```

- 1- Yeni verisetindeki son 100 veriyi TestData'ya eşitliyoruz.
- 2- Yeni veri setinde İloc kullanarak, sütunları 0 ve 1 den yani ilk 2 sütunu seçtik ve Training_Set e eşitledik.
- 3- Tüm satırları seçeceğimiz için satır kısmı boş kaldı(virgülden önceki kısım).
- 4- Eşitlediğimiz Training_Set'ten son 60 veriyi çıkarttık.

```
In [18]: print("Training Set Shape ", Training_Set.shape)  
print("Test Set Shape ", TestData.shape)
```

```
Training Set Shape (4995, 1)  
Test Set Shape (100, 4)
```

- 1- Training_Set'de kaç tane veri olduğunu ekrana yazdırdık.
- 2- Test_Data'da kaç tane veri olduğunu ekrana yazdırdık.

```
In [19]: #Training_Set'in ilk 10 verisini head metodu ile ekrana yansıttık.  
Training_Set.head(10)
```

Out[19]:

MW	
Datetime	
2004-10-01	1924.130435
2004-10-02	1641.583333
2004-10-03	1528.708333
2004-10-04	1926.541667
2004-10-05	1970.125000
2004-10-06	1982.458333
2004-10-07	1975.583333
2004-10-08	1962.625000
2004-10-09	1681.250000
2004-10-10	1551.875000

```
In [20]: #Training_Set'in son 10 verisini tail metodu ile ekrana yansıttık.  
Training_Set.tail(10)
```

Out[20]:

MW	
Datetime	
2018-05-26	1827.708333
2018-05-27	1940.541667
2018-05-28	2112.208333
2018-05-29	2443.750000
2018-05-30	2453.000000
2018-05-31	2412.916667
2018-06-01	2204.208333
2018-06-02	1962.708333
2018-06-03	1854.375000
2018-06-04	1958.375000

```
In [21]: #TestData'nın ilk 10 verisini head metodu ile ekrana yansıttık.  
TestData.head(10)
```

Out[21]:

MW				
Ay				
Yil				
Hafta				
Datetime				
2018-04-26	1806.666667	4	2018	17
2018-04-27	1721.375000	4	2018	17
2018-04-28	1578.083333	4	2018	17
2018-04-29	1552.291667	4	2018	17
2018-04-30	1814.333333	4	2018	18
2018-05-01	1819.083333	5	2018	18
2018-05-02	1923.958333	5	2018	18
2018-05-03	1978.041667	5	2018	18
2018-05-04	1916.125000	5	2018	18
2018-05-05	1515.916667	5	2018	18

```
In [22]: type(Training_Set.values)
```

Out[22]: numpy.ndarray

1- Training_Set'in türünü öğrenmek için yazdık.

```
In [23]: sc = MinMaxScaler(feature_range=(0, 1))  
Train = sc.fit_transform(Training_Set)
```

- 1- Eğitim setinde verilen aralıkta, yani sıfır ile bir arasında olacak şekilde her özelliği ayrı ayrı ölçeklendirdim.
- 2- Eğitim setine parametrelerin uydurulması için fit_transform() methodunu kullandık

```
In [24]: X_Train = []
Y_Train = []

for i in range(60, Train.shape[0]):

    X_Train.append(Train[i-60:i])
    Y_Train.append(Train[i])

X_Train = np.array(X_Train)
Y_Train = np.array(Y_Train)

print(X_Train.shape)
print(Y_Train.shape)
```

```
(4935, 60, 1)
(4935, 1)
```

- 1- X_Train ve Y_Train adında 2 dizi belirledik.
- 2- Dizi Aralığımız 60 Değerden SONSUZ'a kadar olmalıdır.
- 3- X_Train 0-59 Arasında
- 4- Y, geçmiş 60 Değerine göre 60. Değer olacaktır.
- 5- Numpy Array'e Dönüştürdük.
- 6- X_Train ve Y_Train dizisinde kaç tane veri(eleman) sayısı olduğunu ekrana yazdırdık.

```
In [25]: X_Train = np.reshape(X_Train, newshape=(X_Train.shape[0], X_Train.shape[1], 1))
X_Train.shape
```

```
Out[25]: (4935, 60, 1)
```

- 1- Şekil (Veri Noktası(Datapoints),Adımlar(Steps), 1) sayısı şeklinde olmalıdır
- 2- 3-d Vektör veya #rd Vektör Boyutlara dönüştürüyoruz.
- 3- reshape komutu ile ekrana yazdırdık.

```
In [26]: #Gerekli Keras kütüphanelerini ve tensorflow kütüphanesini yazıyoruz.
from keras.models import Sequential
from keras.layers import Dense,LSTM
from tensorflow.keras.layers import Dense, Dropout
```

```
In [27]: regressor = Sequential()

regressor.add(LSTM(units = 50, return_sequences = True, input_shape = (X_Train.shape[1], 1)))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return_sequences = True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))
regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

regressor.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

- 1- regresörümüz ardışık olacağından Sequential'e eşitliyoruz.
- 2- İlk LSTM katmanını ve bazı Bırakma(Dropout) düzenlemelerini ekleme.
- 3- İkinci bir LSTM katmanı ve bir miktar Bırakma(Dropout) regülasyonu ekleme.
- 4- Üçüncü bir LSTM katmanı ve bir miktar Bırakma(Dropout) regülasyonu ekleme.
- 5- Dördüncü bir LSTM katmanı ve bazı Bırakma(Dropout) düzenlemelerinin eklenmesi.
- 6- Çıktı katmanını ekledik.
- 7- RNN'yi derledik.

```
In [28]: history=regressor.fit(X_Train, Y_Train, epochs = 30, batch_size = 32)

Epoch 1/30
155/155 [=====] - 9s 57ms/step - loss: 0.0258
Epoch 2/30
155/155 [=====] - 9s 57ms/step - loss: 0.0214
Epoch 3/30
155/155 [=====] - 9s 58ms/step - loss: 0.0200
Epoch 4/30
155/155 [=====] - 9s 56ms/step - loss: 0.0193
Epoch 5/30
155/155 [=====] - 9s 56ms/step - loss: 0.0192
Epoch 6/30
```

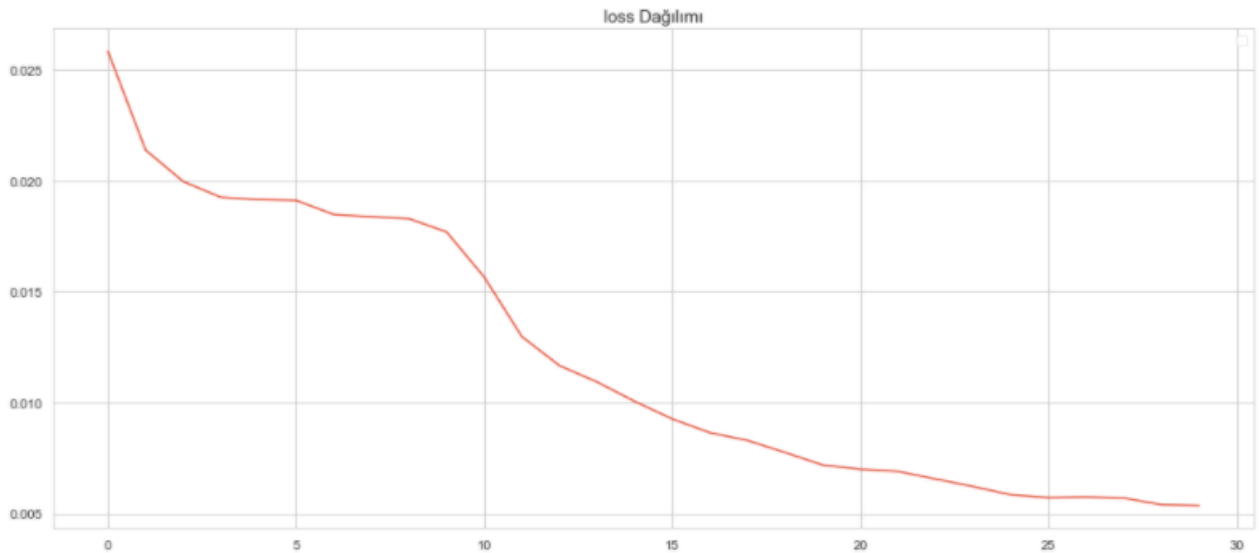
- 1- Epoch, bir TÜM veri kümesinin yalnızca BİR KEZ sınır ağından ileri ve geri aktarılmasıdır. Biz 30 kez ileri geri aktardık.
- 2- batch_size, öğrenme sürecinde ağı eğitmek için kullanılacak olan eğitim örneğinizin alt küme boyutunu belirtir.
- 3- Bu işlemi history'e eşitledik çünkü loss grafiği için önemli.

```
In [29]: #İşlem sonunda History'deki tüm veri kümelerini listeledik
print(history.history.keys())

dict_keys(['loss'])
```

```
In [30]: plt.plot(history.history['loss'])
plt.title("loss Dağılımı")
plt.legend()
No handles with labels found to put in legend.
```

```
Out[30]: <matplotlib.legend.Legend at 0x2cf7a9fc580>
```



- 1- 'loss' grafiği için history içindeki loss(kayıp) bölümü burda belirttik
- 2- Grafiğe loss Dağılımı adında başlık atıyoruz.
- 3- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.

```
In [31]: #TestDataındaki 100 veriden son 10 veriyi ekrana yazdırdık.Bunun için tail() metodunu kullandık.
TestData.tail(10)
```

```
Out[31]:
```

	MW	Ay	Yil	Hafta
Datetime				
2018-07-25	2321.166667	7	2018	30
2018-07-26	2300.333333	7	2018	30
2018-07-27	2126.541667	7	2018	30
2018-07-28	1810.083333	7	2018	30
2018-07-29	1738.791667	7	2018	30
2018-07-30	2026.333333	7	2018	31
2018-07-31	2051.083333	7	2018	31
2018-08-01	2060.291667	8	2018	31
2018-08-02	2168.208333	8	2018	31
2018-08-03	2042.000000	8	2018	31

```
In [32]: TestData.shape
```

```
Out[32]: (100, 4)
```

1- TestData'da kaç tane veri sayısının olduğunu ekrana yazdırdık.

```
In [33]: NewDataSet.shape
```

```
Out[33]: (5055, 4)
```

1- NewDataSet'de kaç tane veri sayısının olduğunu ekrana yazdırdık.

```
In [34]: Df_Total = pd.concat((NewDataSet[["MW"]], TestData[["MW"]]), axis=0)
```

1- concat metodu ile farklı verileri birleştirdik ve bunu Df_Total'a eşitledik.

```
In [35]: Df_Total.shape
```

```
Out[35]: (5155, 1)
```

1- Veri setinde kaç tane veri olduğunu shape metodu ile ekrana yansıttık.

```
In [36]: inputs = Df_Total[len(Df_Total) - len(TestData) - 60:].values  
inputs.shape
```

```
Out[36]: (160, 1)
```

1- len() Fonksiyon bir nesnenin öge sayısını döndürdüğü için ayırdığımız verilerdeki öge sayısını döndürdük.

```
In [37]: inputs = Df_Total[len(Df_Total) - len(TestData) - 60:].values

inputs = inputs.reshape(-1,1)

inputs = sc.transform(inputs)

X_test = []
for i in range(60, 160):
    X_test.append(inputs[i-60:i])

X_test = np.array(X_test)

X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

Tahmini_tuketim = regressor.predict(X_test)
Tahmini_tuketim = sc.inverse_transform(Tahmini_tuketim)
```

- 1- Yeniden şekillendirmemiz gerekiyor bunun için reshape metodunu kullandık.
- 2- Veri Kümesini Normalleştirmek için transform metodunu kullandık.
- 3- 60 ile 160 arasındaki değerlere bakıyoruz.
- 4- Numpy Array'e Dönüştürdük.
- 5- Ağa Geçmeden Önce Yeniden Şekillendirmek için reshape metodunu kullandık.
- 6- Modele Geçiş kodunu yazdık.
- 7- Değerleri elde etmek için ters Dönüşüm yaptık.

```
In [38]: Gercek_MegaWatt = TestData["MW"].to_list()
Tahmin_MegaWatt = Tahmini_tuketim
dates = TestData.index.to_list()
```

- 1- TestData listesindeki "MW" değerlerini Gerçek_MegaWatt'a eşitledik bu işlemi grafiğe dökmek için kullandık.
- 2- Oluşturduğumuz Model çıktısındaki değerleri Tahmin_MegaWatt'a eşitledik.
- 3- TestDatasını içindeki ilk index olan Datetime Kısmının listesini dates kısmına eşitledik.


```
In [39]: Machine_Df = pd.DataFrame(data={
    "Date":dates,
    "Gercek MegaWatt": Gercek_MegaWatt,
    "Tahmin MegaWatt":[x[0] for x in Tahmin_MegaWatt ]
})
```

- 1- Machine_Df adında bir Veri kümesi oluşturup Sutunlara tarih,gercek MegaWatt ve Tahmin MegaWatt değerlerini ekleyip
- 2- Satırlarada karşılık gelen değerlerini ekliyoruz.

```
In [40]: #Machine_Df veri kümesini ekrana yansıtıyoruz.
Machine_Df
```

Out[40]:

	Date	Gercek MegaWatt	Tahmin MegaWatt
0	2018-04-26	1806.666667	1818.164429
1	2018-04-27	1721.375000	1801.282349
2	2018-04-28	1578.083333	2021.361938
3	2018-04-29	1552.291667	1784.335205
4	2018-04-30	1814.333333	1699.168579
...
95	2018-07-30	2026.333333	2053.016357
96	2018-07-31	2051.083333	2081.938721
97	2018-08-01	2060.291667	2069.596680
98	2018-08-02	2168.208333	2078.161133
99	2018-08-03	2042.000000	2094.181885

100 rows x 3 columns

```
In [41]: Gercek_MegaWatt = TestData["MW"].to_list()
Tahmin_MegaWatt = [x[0] for x in Tahmin_MegaWatt ]
dates = TestData.index.to_list()
```

- 1- TestData listesindeki "MW" değerlerini Gerçek_MegaWatt'a eşitledik bu işlemi grafiğe dökmek için kullandık.
- 2- Oluşturduğumuz Model çıktısındaki değerler dizisini for döngüsü ile Tahmin_MegaWatt'a eşitledik.
- 3- TestDatasını içindeki index olan Datetime Kısmının listesini dates kısmına eşitledik.

Tahmin Modeli Grafiği

```
In [42]: fig = plt.figure()
ax1= fig.add_subplot(111)

style.use('ggplot')
plt.title('Tahmin Modeli')

x = dates
y1 = Tahmin_MegaWatt
sns.lineplot(x,y1, color="red")
sns.set(rc={'figure.figsize':(15,6)})
plt.gcf().autofmt_xdate()

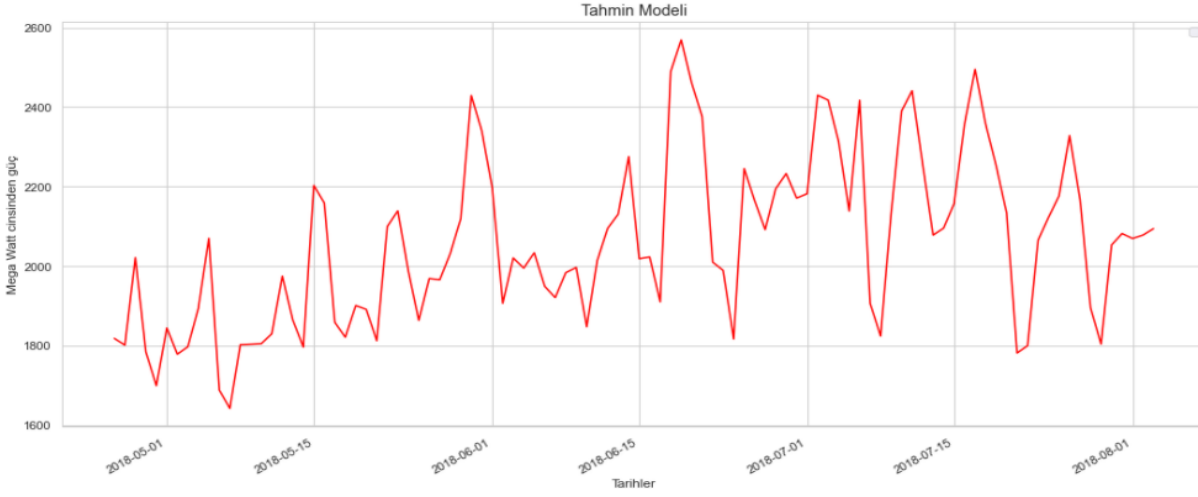
plt.xlabel('Tarihler')
plt.ylabel("Mega Watt cinsinden güç")

plt.legend()

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

- 1- plt.figure() ile 'fig' adında bir figür oluşturduk.
- 2- fig.add_subplot() komutu ile ax1 eksenini ekledik.
- 3- R dili görselleştirmesi olan ggplot'u style.use('ggplot') diyerek ggplot stilini kullandık.
- 4- Tahmin Modeli adında başlık attık bunun için title kullandık.
- 5- Grafiğimizde x koordinatı için yukarıda belirlediğimiz dates'i y koordinatı için ise Tahmin_MegaWatt'ı kullandık.
- 6- sns.lineplot ile seaborn kütüphanesinden çizgi grafiğini kullandık.
- 7- Bu grafiğin x ve y koordinatlarını belirlemek için x değerine yukarıda eşitlediğimiz x değerini yazdırdık
- 8- y koordinatına da yukarıda eşitlediğimiz y1 değerini yazdırdık ve çizgi renginde kırmızı olarak belirledik.
- 9- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- 10- x etiketlerini güzelleştirelim bu kod bize tarihlerin eğik bir şekilde durmasını sağlayacaktır.
- 11- x koordinatı için Tarihler adında etiket belirledik.
- 12- y koordinatı için Mega Watt cinsinden güç adında etiket belirledik.
- 13- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.

Çıktısı;



Gerçek Değer Modeli Grafiği

```
In [43]: fig = plt.figure()
ax1= fig.add_subplot(111)
plt.title('Gerçek değer Modeli')

style.use('ggplot')
x = dates
y = Gercek_MegaWatt

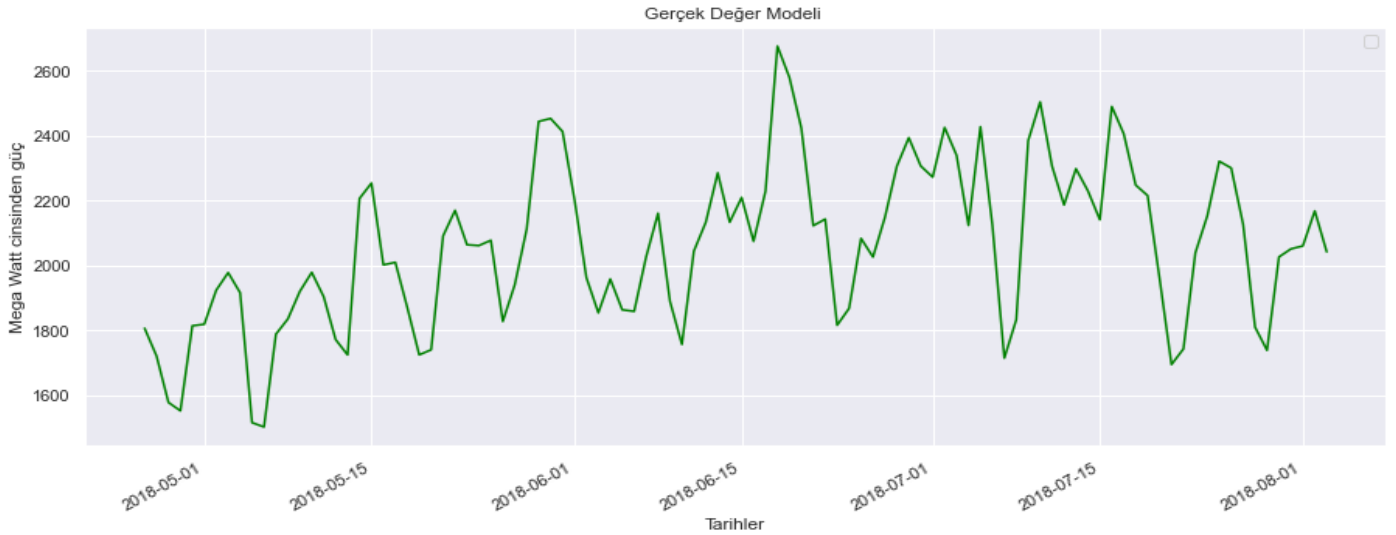
sns.lineplot(x,y, color="green")
sns.set(rc={'figure.figsize':(15,6)})
plt.gcf().autofmt_xdate()

plt.xlabel('Tarihler')
plt.ylabel("Mega Watt cinsinden güç")
plt.title("Gerçek Değer Modeli ")
plt.legend()
```

- 1- plt.figure() ile 'fig' adında bir figür oluşturduk.
- 2- fig.add_subplot() komutu ile ax1 eksenini ekledik.
- 3- Gerçek değer Modeli adında başlık attık grafiğe.
- 4- R dili görselleştirmesi olan ggplot'u style.use('ggplot') diyerek ggplot stilini kullandık.
- 5- sns.lineplot ile seaborn kütüphanesinden çizgi grafiğini kullandık.
- 6- Bu grafiğin x ve y koordinatlarını belirlemek için x değerine yukarıda eşitlediğimiz x değerini yazdırdık
- 7- y koordinatına da yukarıda eşitlediğimiz y değerini yazdırdık ve çizgi rengini de yeşil olarak belirledik.

- 8- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- 9- x etiketlerini güzelleştirelim bu kod bize tarihlerin eğik bir şekilde durmasını sağlayacaktır.
- 10- x koordinatına tarihler adında etiket ekledik.
- 11- y koordinatına Mega Watt cinsinden güç adında etiket ekledik.
- 12- Grafiğe Gerçek Değer Modeli adında Başlık ekledik bunun için title metodunu kullandık.
- 13- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.

Çıktısı;



Tahmin Model Grafiği ile Gerçek Değer Model Grafiği Karşılaştırılması

```
In [44]: fig = plt.figure()
ax1= fig.add_subplot(111)
plt.title('Gerçek değer ile Tahmin Modelinin karşılaştırılması')
style.use('ggplot')

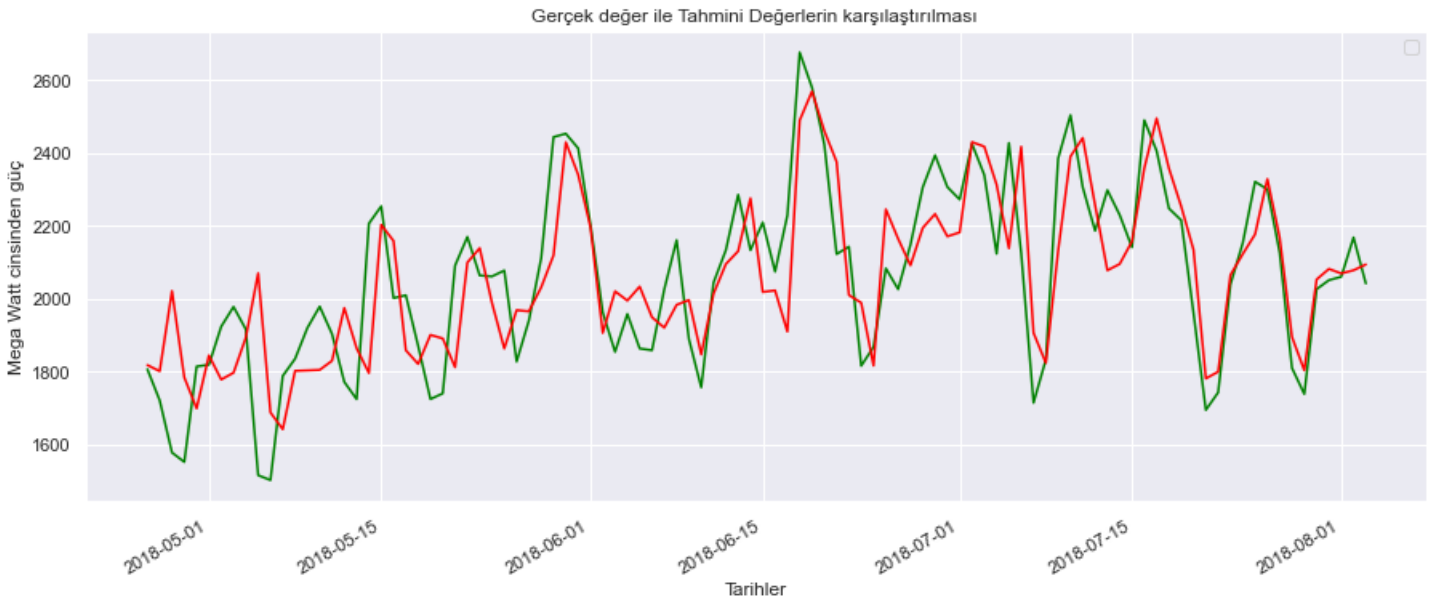
x = dates
y = Gercek_MegaWatt
y1 = Tahmin_MegaWatt

sns.lineplot(x,y, color="green")
sns.lineplot(x,y1, color="red")
sns.set(rc={'figure.figsize':(15,6)})
plt.gcf().autofmt_xdate()

plt.xlabel('Tarihler')
plt.ylabel("Mega Watt cinsinden güç")
plt.title("Gerçek değer ile Tahmini Değerlerin karşılaştırılması ")
plt.legend()
```

- 1- plt.figure() ile 'fig' adında bir figür oluşturduk.
- 2- fig.add_subplot() komutu ile ax1 eksenini ekledik.
- 3- Gerçek değer ile Tahmin Modelinin karşılaştırılması adında grafik için başlık oluşturduk.
- 4- R dili görselleştirmesi olan ggplot'u style.use('ggplot') diyerek ggplot stilini kullandık.
- 5- sns.lineplot ile seaborn kütüphanesinden çizgi grafiğini kullandık.
- 6- Bu grafiğin x ve y koordinatlarını belirlemek için x değerine yukarıda eşitlediğimiz x değerini yazdırdık
- 7- Gerçek Değerler için y koordinatına da yukarıda eşitlediğimiz y değerini yazdırdık.
- 8- Tahmin modeli için y koordinatına da yukarıda eşitlediğimiz y1 değerini yazdırdık.
- 9- Gerçek değerlerin grafik çizgi rengini kırmızı,tahmini değerlerin rengini yeşil ayarladık.
- 10- Bunu yapmak karşılaştırmada aradaki farkı daha net görmemizi sağlayacaktır.
- 11- Daha sonra sns.set ile grafiğin en boy oranını belirledik.
- 12- x etiketlerini güzelleştirelim bu kod bize tarihlerin eğik bir şekilde durmasını sağlayacaktır.
- 13- x koordinatına Tarihler adında etiket atadık.
- 14- y koordinatına Mega Watt cinsinden güç adında etiket atadık.
- 15- Grafiğe başlık olarak "Gerçek değer ile Tahmini Değerlerin karşılaştırılması " yazdırdık.
- 16- plt.legend() diyerek grafiğimizin otomatik olarak oluşturulmasını sağladık.

Çıktısı;



4. SONUÇ

Şimdiye kadar yapılan işlemlerde öncelikle veri seti incelendi daha sonra Jupyter Notebook ortamında kodlamaya başlandı. Gerekli kütüphaneler kod bölümüne yazıldı ve veri seti hazırlanıp ekrana veri seti ile ilgili bilgileri yazdırıldı. Zaman serili veri setimizde Tarih ve Saat sütunları yeniden biçimlendirildi. Daha sonra veri setinde benzersiz kaç tane yıl ve ay olduğunu bulunmuş oldu. Diğer adımda da Grafik modellerine geçiş yapıp grafikler hakkında bilgi verilip yapılan grafik modellerini veri setimize uyarlamak için gerekli tüm grafik kod açıklamaları yazıldı ve çıktıları gösterildi. Veri setini günlere göre ayarlanıp oluşan veri den son 100 veri çıkartılıp Yeni oluşan veri seti LSTM katmanları oluşturularak Epoch'unda yardımıyla makine eğitildi ve yeni bir tahmini 100 günlük model oluşturuldu. Daha sonra çıkardığımız veri setinden çıkardığımız son 100 veri ile tahmin modelinin grafiği tek tek ekrana yansıtıldı. Bu iki grafiği daha sonra aynı grafiğe koyarak karşılaştırma yapıldı.

5. KAYNAKLAR

<https://www.mathworks.com/discovery/deep-learning.html>

<https://machinelearningmastery.com/findings-comparing-classical-and-machine-learning-methods-for-time-series-forecasting/>

<https://machinelearningmastery.com/what-is-deep-learning/>

<https://towardsdatascience.com/what-is-a-data-set-9c6e38d33198>

https://www.tutorialspoint.com/python_pandas/index.htm

<https://www.unalzafer.com/python-veri-bilimi-icin-pandas/>

https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.html

<https://machinelearningmastery.com/seaborn-data-visualization-for-machine-learning/>

<https://arxiv.org/pdf/1909.13316v1.pdf>

<https://machinelearningmastery.com/what-is-deep-learning/>

<https://machinelearningmastery.com/multi-step-time-series-forecasting-with-machine-learning-models-for-household-electricity-consumption/>

https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.html

<https://www.geeksforgeeks.org/python-data-analysis-using-pandas/>

<https://matplotlib.org/tutorials/introductory/pyplot.html>

<https://www.veribilimiokulu.com/blog/seaborn-ile-veri-gorsellestirmesi/>

<https://towardsdatascience.com/seaborn-python-8563c3d0ad41>

<https://python-graph-gallery.com/seaborn/>

<https://svcmistry.org/tr/dictionary/what-is-the-difference-between-matplotlib-and-seaborn-which-one-should-i-learn-for-studying-data-science/>

<https://pymotw.com/2/pprint/>

<https://medium.com/better-programming/how-to-pretty-print-in-python-9b1d8764d151>

<https://www.geeksforgeeks.org/pprint-data-pretty-printer-python/>

<https://medium.com/bili%C5%9Fim-hareketi/veri-bilimi-i%C3%A7in-temel-python-k%C3%BCt%C3%BCphaneleri-2-pandas-dcc12ae01b7d>

<http://www.veridefteri.com/2018/03/30/pandasa-giris-veri-cerceveleri/>

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://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