

HİPERPARAMETRE AYARLI DERİN ÖĞRENME YÖNTEMLERİ İLE ELEKTRİK TÜKETİMİNİN TAHMINİ

DENİZ CAN TOŞUR

BURDUR,2021

ÖZET

Bu rapor çalışmasında, 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. LSTM Modeli için Multivariable(Çok değişkenli) veri seti kullanılmıştır. Öncelikle modeldede önce kütüphaneler belirlendi daha sonra veri seti ekrana yansıtılıp veri setini işledikten(düzenledikten) sonra grafikler ile güzelleştirme yapılp modeller oluşturuldu. LSTM modelinde gerçek değer ile LSTM 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 model geliştirildi. Daha sonra ekleme yapılarak jackknife ve bootstrap ile veri arttırma teknikleri kullanıldı. Modelin eğitimi tamamlandıktan sonra k katlamalı çapraz doğrulama(k fold cross validation) ile birlikte GridSearchCV ile doğruluk değerlerini ve tahmin edicileri bulduk.

Anahtar Kelimeler: lineplot, boxplot, distplot, violinplot, LSTM, Jackknife, Bootstrap, Kfold Cross Validation, GridSearchCV

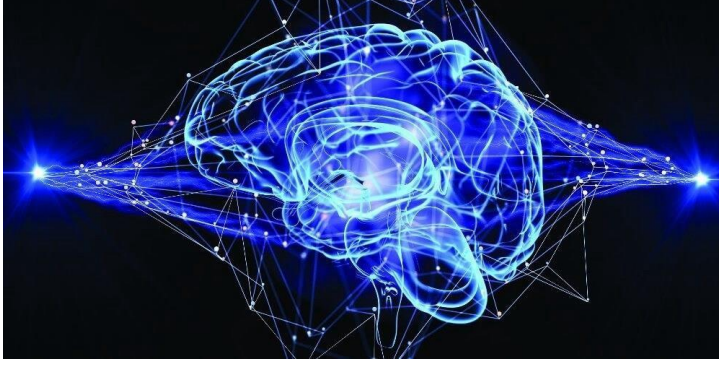
ABSTRACT

In this report, between 2006-2011 with multivariate (multivariate) input, firstly, by showing the electricity consumption with 'Jupyter Notebook', data visualization, beautification and arrangement was made in this data area. Multivariate (Multivariate) data set is used for LSTM Model. First, libraries were determined in the model, then the models were created after the data set screen was illuminated and the data set was processed (edited) and the models were beautified with graphics. In the LSTM model, the real value and the LSTM model estimation were compared. Functions using the method of displaying on the chart The data set was converted into the charts using the functions 'lineplot' for line chart, 'boxplot' for box chart, 'distplot' for histogram chart, 'violin chart' for Violin Chart. Later, data pre-processing can be continued model was developed. Then by adding jackknife and bootstrap data increase style query. After completing the model training, we found accuracy pasting and predictors with GridSearchCV along with k-coded validation (cross validation).

Keywords: lineplot, boxplot, distplot, violinplot, LSTM, Jackknife, Bootstrap, Kfold Cross Validation, GridSearchCV

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 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 seti 2006-2011 yılları arasında enerji tüketimini gösteren veri seti ele alınmıştır. Veri seti Çok değişkenli veri seti olarak 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.

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	2006-12-16,1209.1760000000004,34.921999999999976,93552.530000000006,5180.800000000008,0.0,546.0,4926.0,14680.933319299998
3	2006-12-17,3390.46,226.0059999999994,345725.32000000024,14398.599999999998,2033.0,4187.0,13341.0,36946.66673200004
4	2006-12-18,2203.825999999997,161.7919999999966,347373.64000000036,9247.199999999988,1063.0,2621.0,14018.0,19028.43328100003
5	2006-12-19,1666.1940000000006,150.9419999999978,348479.0100000004,7094.00000000014,839.0,7602.0,6197.0,13131.900043499994

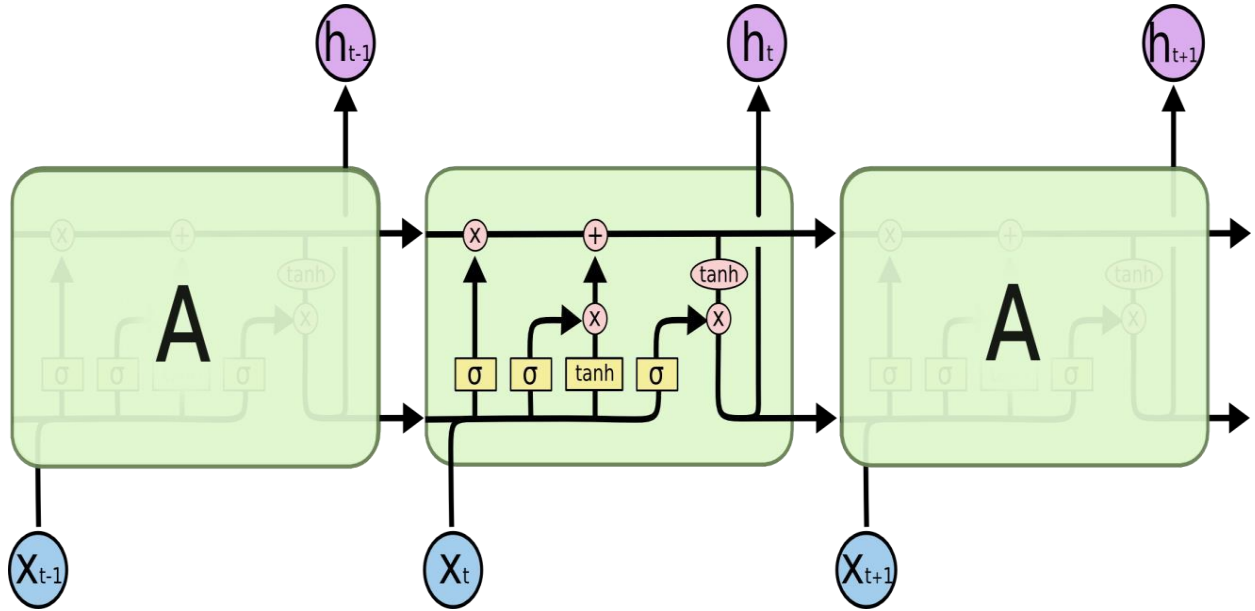
Veriler aşağıya doğru devam etmektedir.

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 davranışdır, öğrenmek için mücadele ettikleri bir şey değildir.

Örnek bir LSTM modeli aşağıdaki gibi çalışır.



3.3.1 Kütüphaneler ve Veri seti İşlemleri

```
In [2]: 1 # Kütüphaneleri import ediyoruz.
2
3 import numpy as np
4 np.random.seed(1)
5 import tensorflow
6 tensorflow.random.set_seed(2)
7 import pandas as pd
8 import matplotlib.pyplot as plt
9 from keras.models import Sequential, load_model
10 from keras.layers.core import Dense
11 from keras.layers.recurrent import LSTM
12 from keras import optimizers
13 from keras.callbacks import EarlyStopping
14 from sklearn.preprocessing import MinMaxScaler
15 from sklearn.metrics import mean_squared_error, r2_score
16 from math import sqrt
17 import datetime as dt
18 import time
19 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]: 1 # Erken durdurma kurma
2 earllystop = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=80, verbose=1, mode='min')
3 callbacks_list = [earllystop]
```

- Öncelikle earllystop(Erken durdurma) metodunu uyguluyoruz.
- Bu işlemi eğer LSTM modelinin doğrulama veri kümesindeki performansı azalmaya başlarsa bu sayede earllystop ile erken durduma işlemi yapılacaktır.
- Daha sonra earllystop'u callbacks_list'e eşitliyoruz.

Veri Ön İşleme

```
In [3]: 1 df = pd.read_csv("household_daily.csv",header=0,infer_datetime_format=True)
2 df.head()
```

Out[3]:

	datetime	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_m
0	2006-12-16	1209.176	34.922	93552.53	5180.8	0.0	546.0	
1	2006-12-17	3390.460	226.006	345725.32	14398.6	2033.0	4187.0	
2	2006-12-18	2203.826	161.792	347373.64	9247.2	1063.0	2621.0	
3	2006-12-19	1666.194	150.942	348479.01	7094.0	839.0	7602.0	
4	2006-12-20	2225.748	160.998	348923.61	9313.0	0.0	2648.0	

- 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]: 1 # Yılın Ayı, Günü, Saati vb. Gibi Tüm Verileri Çıkartıyoruz
2
3 df["Ay"] = pd.to_datetime(df["datetime"]).dt.month
4 df["Yıl"] = pd.to_datetime(df["datetime"]).dt.year
5 df["Tarih"] = pd.to_datetime(df["datetime"]).dt.date
6 df["Saat"] = pd.to_datetime(df["datetime"]).dt.time
7 df["Hafta"] = pd.to_datetime(df["datetime"]).dt.week
8 df["Gün"] = pd.to_datetime(df["datetime"]).dt.day_name()
9 df = df.set_index("datetime")
10 df.index = pd.to_datetime(df.index)
11 df.head(5)
```

- Öncelikle veri setimizi kısıltması 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 içerisinde başlığını yazıp(Yıl) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'year' yani yıl verilerini buraya yazmasını istedik.
- dataset içerisinde başlığını yazıp(Tarih) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'date' yani tarih verilerini buraya yazmasını istedik.
- dataset içerisinde başlığını yazıp(Saat) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'time' yani saat verilerini buraya yazmasını istedik.
- dataset içerisinde başlığını yazıp(Hafta) hangi türden veri çekeceğini yazıyoruz burdaki işlemde 'week' yani hafta verilerini buraya yazmasını istedik.
- dataset içerisinde başlığını 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;

	Global_active_power	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	Sub_metering_2	Sub_metering_3	Sub_metering_4	Ay	Y
datetime										
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

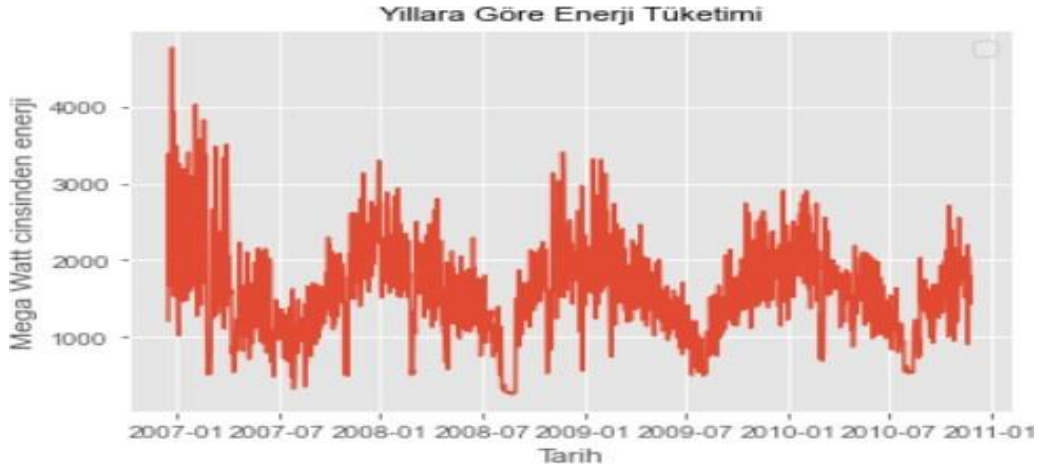
Çizgi Grafiği

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

```
In [5]: 1 #Çizgi Grafiği
2 sns.lineplot(x=df["Tarih"], y=df["Global_active_power"], data=df)
3 sns.set(rc={'figure.figsize':(15,6)})
4 plt.xlabel("Tarih")
5 plt.ylabel("Mega Watt cinsinden enerji")
6 plt.legend()
7 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 setimizden 'Tarih' kısmını yazdırdık, y değerine de veri setimizden '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 merkezsiz konum, yayılma, çarpıklık ve basıklık yönünden verileri özetlemek ve aykırı değerleri tanımlamak için kullanılır.

```
In [6]: 1 #Kutu Grafiği
2 sns.boxplot(x=df["Yil"], y=df["Global_active_power"], data=df)
3 sns.set(rc={'figure.figsize':(15,6)})
4 plt.xlabel("Tarih")
5 plt.ylabel("Mega Watt cinsinden enerji")
6 plt.legend()
7 plt.title("Yıllara Göre Enerji Tüketimi")
```

- sns.boxplot ile seaborn kütüphanesinden kutu grafiğini kullandık. Bu grafiğin x ve y koordinatlarını belirlemek için x değerine veri setimizden 'Yil' kısmını yazdırdık, y koordinatına da veri setimizden '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 altına 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 'Yıllara Göre Enerji Tüketimi' yazdırıyoruz.

Çıktısı;



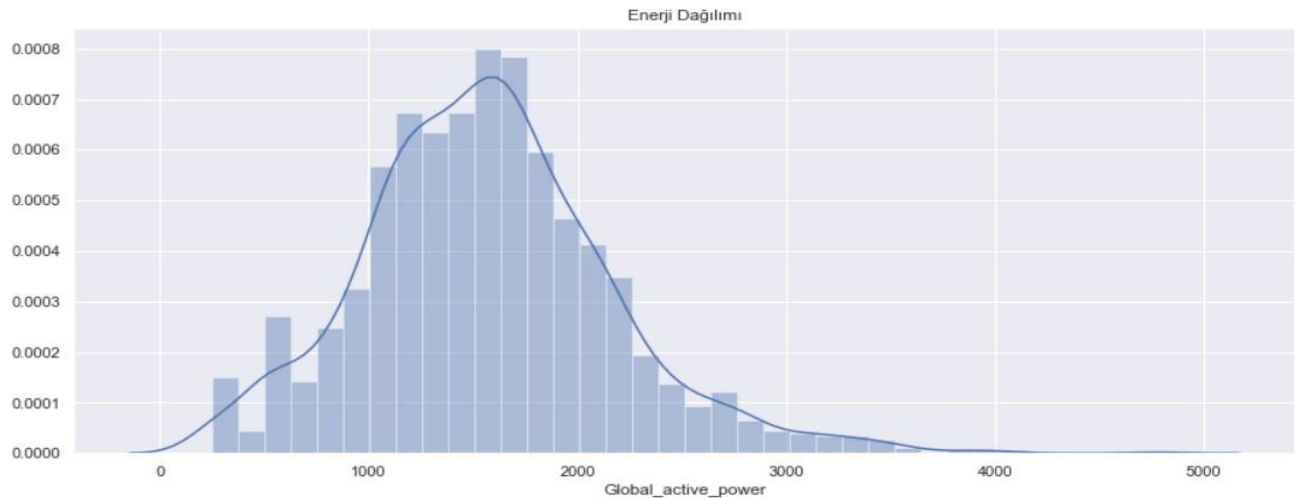
Histogram Grafiği

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]: 1 #Histogram Grafiği
2 sns.distplot(df["Global_active_power"])
3 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;



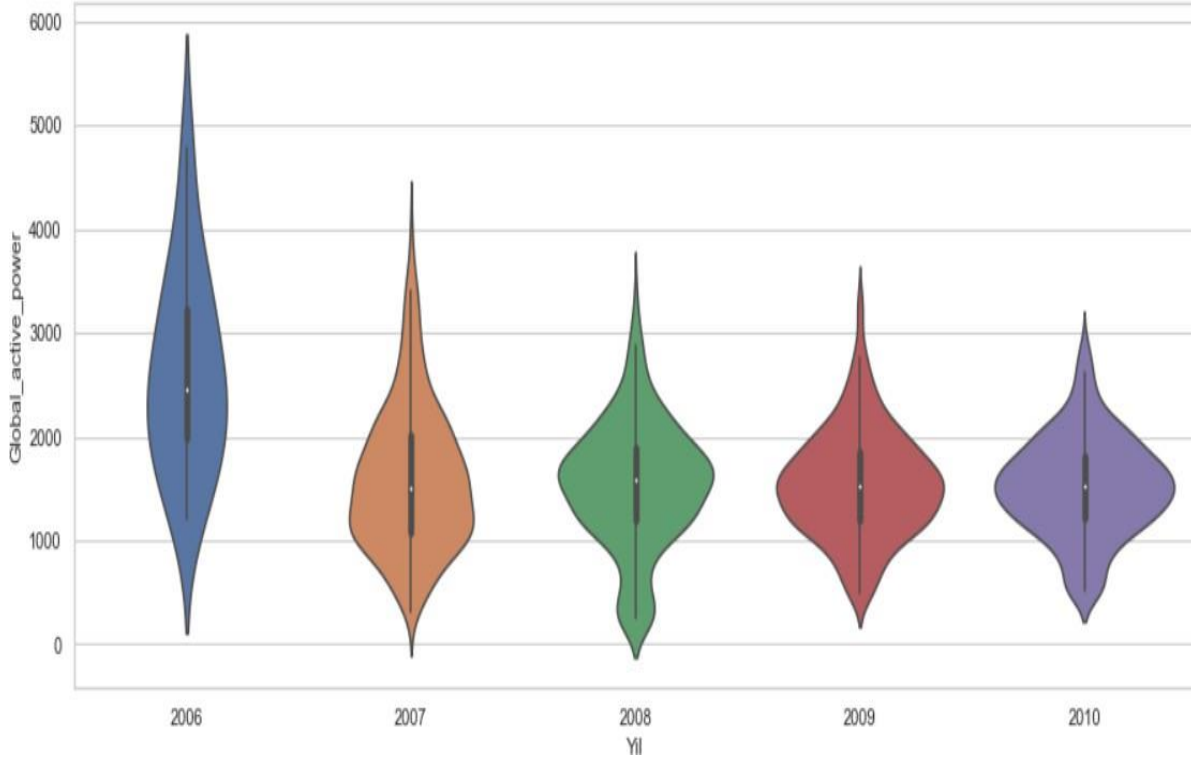
Violin(Keman) Grafiđi

Bir keman grafiđi , sayısal verileri çizme yöntemidir. Her bir tarafa döndürölmüş bir çekirdek yoğunluđu grafiđinin eklenmesiyle bir kutu grafiđine benzer . Keman grafikleri , genellikle bir çekirdek yoğunluđu tahmincisi tarafından düzleştirilen farklı değerde verilerin olasılık yoğunluđunu göstermeleri dışında kutu grafiklerine benzer. Tipik olarak bir keman grafiđi, bir kutu grafiđindeki tüm verileri içerir. Bir keman grafiđinin birden fazla katmanı olabilir. Örneđin, dış şekil tüm olası sonuçları temsil eder. İçerideki bir sonraki katman, zamanın% 95'inde oluşan değeri temsil edebilir. İçerideki bir sonraki katman (varsa), zamanın% 50'sinde oluşan değeri temsil edebilir.

```
In [8]: 1 #Violin(keman) Grafiđi
        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 grafiđini kullandık. Bu grafiđin x ve y koordinatlarını belirlemek için x değeri veri setimizden 'Yil' kısmını kullandık, y koordinatında veri setimizden 'Global_active_power' kısmını kullanıp yazdırdık.

Çıktısı;



Global_reactive_power'ın Grafikleri

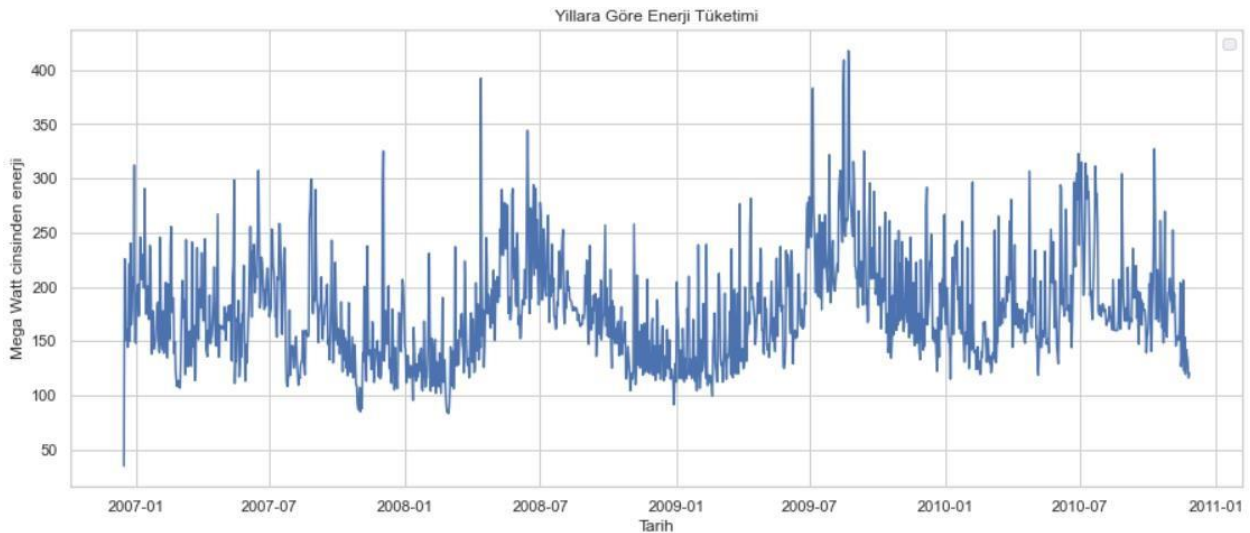
Çizgi Grafiği

```
In [9]: 1 #Çizgi Grafiği
2 sns.lineplot(x=df["Tarih"], y=df["Global_reactive_power"], data=df)
3 sns.set(rc={'figure.figsize':(15,6)})
4 plt.xlabel("Tarih")
5 plt.ylabel("Mega Watt cinsinden enerji")
6 plt.grid(True)
7 plt.legend()
8 plt.title("Yıllara 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 setimizden 'Tarih' kısmını yazdırdık, y değerine de veri setimizden '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(Yıllara Göre Enerji Tüketimi) yazdırdık.

Çıktısı;



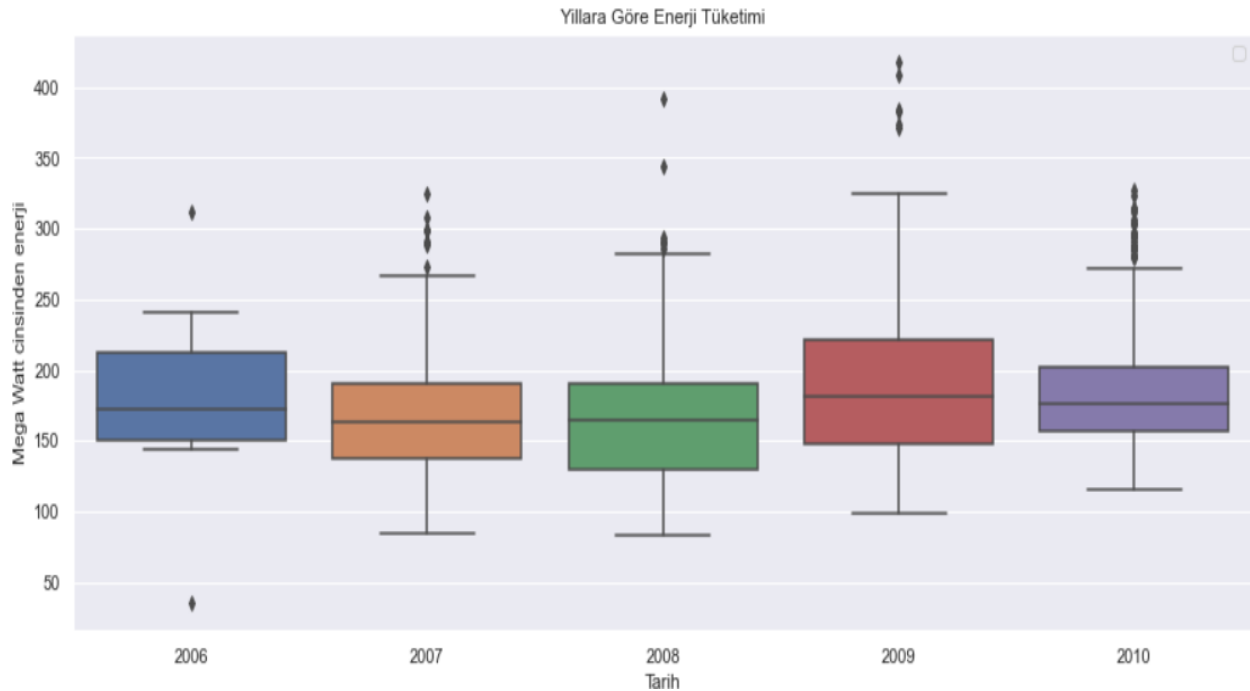
Kutu Grafiği

```
In [10]: 1 #Kutu Grafiği
2 sns.boxplot(x=df["Yil"], y=df["Global_reactive_power"], data=df)
3 sns.set(rc={'figure.figsize':(15,6)})
4 plt.xlabel("Tarih")
5 plt.ylabel("Mega Watt cinsinden enerji")
6 plt.legend()
7 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 setimizden 'Yil' kısmını yazdırdık, y koordinatına da veri setimizden '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 altına 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;

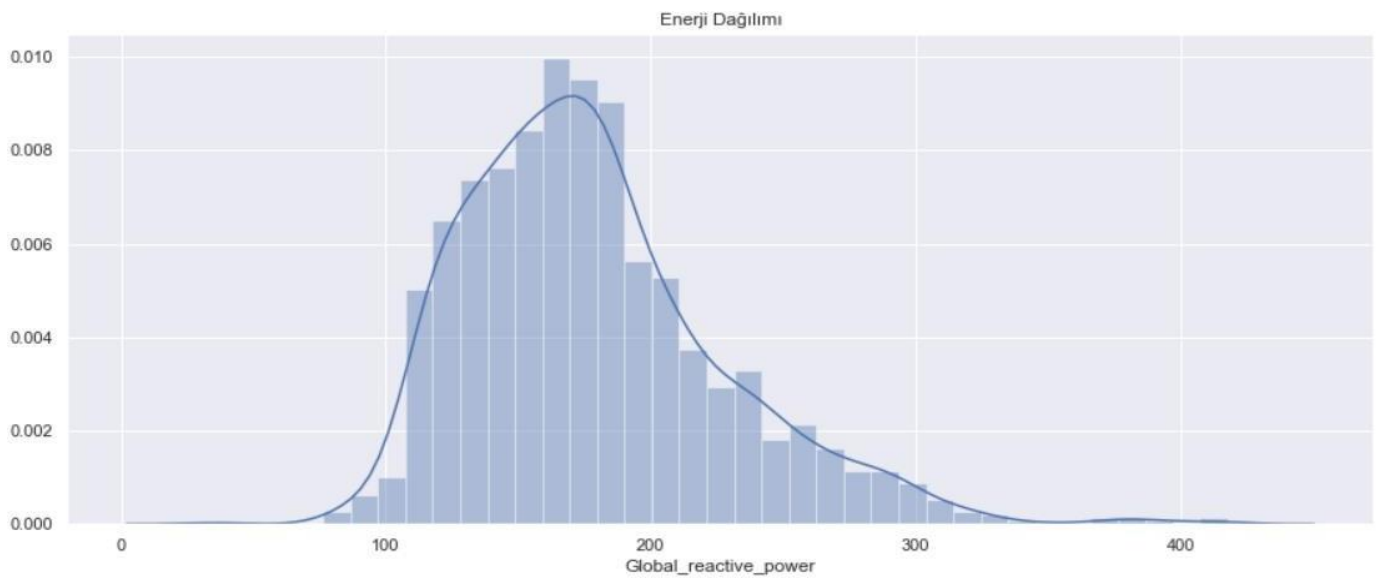


Histogram Grafiği

```
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;

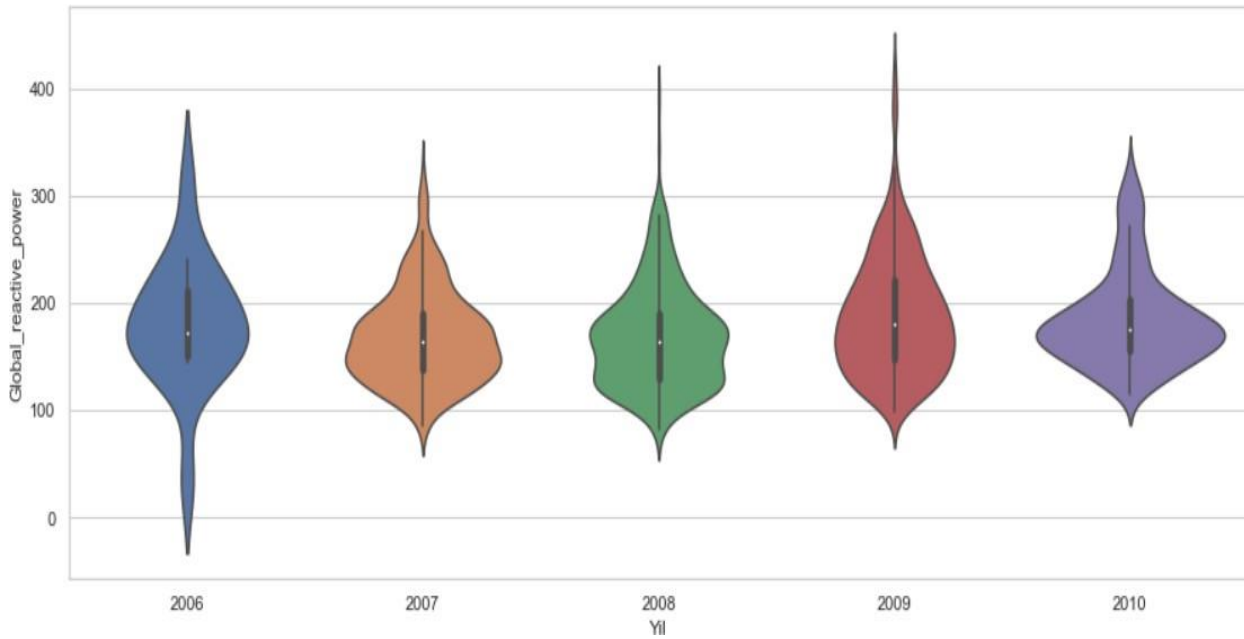


Violin(Keman) Grafiği

```
In [12]: 1 #Violin(keman) Grafiği
          2 sns.set_style('whitegrid')
          3 sns.violinplot(x = 'Yıl', 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 'Yıl' kısmını kullandık, y koordinatınada veri setimizden 'Global_reactive_power' kısmını kullanıp yazdırdık.

Çıktısı;



3.3.3 LSTM 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
Yıl      -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]: 1 print(df.describe().Global_reactive_power)
2 df.drop(df[df['Global_reactive_power']==0].index, inplace = True)
3 #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 istatistiksel 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 [30]: 1 # Erken durdurma kurma
2 earllystop = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=80, verbose=1, mode='min')
3 callbacks_list = [earllystop]
```

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

JACKKNIFE BÖLÜMÜ

```
In [62]: 1 def jackknife(x, func, variance = False):
2         N = len(x)
3         pos = np.arange(N)
4         values = [func(x[pos != i]) for i in pos]
5         jack = np.sum(values)/N
6
7         if variance:
8             values = [np.power(func(x[pos != i]) - jack, 2.0) for i in pos]
9             var = (N-1)/N * np.sum(values)
10            return jack, var
11        else:
12            return jack
```

```
In [63]: 1 x = np.random.normal(0, 2, 100)
2         print(x.std())
3         jackknife(x, np.std, True)
```

1.77031242766317

Out[63]: (1.7701752552752565, 0.016423430647735153)

- Öncelikle def ile jackknife'ı tanımlıyoruz.
- x'in uzunluğunu N'ye eşitliyoruz.
- x'i fonksiyon olarak döngüye sokuyoruz ve bunu valuese eşitliyoruz.
- Np.sum ile oluşan vales'in satır ve sütunlarını toplayıp N'e bölüyoruz ve bunu jack'e eşitliyoruz.
- If koşulu ile varyans diyoruz.
- x fonksiyonunun ilk dizideki dizi ögesi, ikinci ögedeki ögenin gücüne yükseltir ve çıkan sonuçtan jack'i çıkarıp döngüye sokuyoruz.
- (N-1)/N işlemini yapıp valuesin satır ve sütunlarını topladığımız kodu yani np.sum(values) ile çarpıyoruz.
- 2. Kod bloğunda ise rastgele sayı üretiyoruz ve bunu x'e eşitliyoruz.
- Print ile x'in standart sapma sonucunu ve hemen altında jackknife ile jackknife'ın sonucunu ekrana yansıtıyoruz.

BOOTSTRAPPING BÖLÜMÜ

```
In [64]: 1 def bootstrapping(x, n_samples, func=np.mean):
2         y = x.copy()
3         N = len(y)
4         population = []
5
6         for i in range(n_samples):
7             population.append(func(np.random.choice(y, N, replace=True)))
8
9         return np.array(population)
```

- Öncelikle def ile bootstrapping'i tanımlıyoruz.
- X değerini x.copy() ile kopyalayıp y'ye eşitliyoruz.
- len(y) ile y'nin uzunluğunu N'e eşitliyoruz.
- Population dizisi tanımlıyoruz.
- Döngü oluşturup population içerisinde fonksiyon ile y ve N içerisinde rastgele seçilen bir öğeyi döndürmesini sağlıyoruz.

```
In [65]: 1 def histogram(values, n_bins=100):
2         xmax = values.max()
3         xmin = values.min()
4         delta = (xmax-xmin)/n_bins
5
6         counts = np.zeros(n_bins+1, dtype='int')
7
8         for value in values:
9             val_bin = np.around((value-xmin)/delta).astype('int')
10            counts[val_bin] += 1.0
11
12         bins = xmin+delta*np.arange(n_bins+1)
13
14         return bins, counts/values.shape[0]
```

- Histogram'ı tanımlıyoruz.
- xmax bizim values'in max değeri.
- xmin de bizim values'in min değerini temsil ediyor.
- Matematiksel işlem yapıp deltaya eşitliyoruz.
- Sıfırlar ile dolu dizi döndürüp counts'a eşitliyoruz.
- Döngü oluşturup matematiksel ifade yazıyoruz ve onuda np.around ile yuvarlama işlemi yapıp val_bin'e eşitliyoruz.
- np.arange ile dizi oluşturuyoruz ve xmin,delta ile matematiksel işleme sokuyoruz.

```
In [66]: 1 x = np.random.normal(0, 2, size=100)
```

```
In [67]: 1 boot = bootstrapping(x, 1000)
```

```
In [68]: 1 x.mean()
```

```
Out[68]: 0.30558955524039705
```

- Rastgele sayı üretiyoruz ve bunu x'e eşitliyoruz.
- Bootstrapping ile sıradan parametrik olmayan önyüklemeyi uyguladık ve aralığını x ile 1000 arasında seçtik ve boot'a eşitledik.
- X.mean() ile aritmetik ortalamayı hesapladık ve çıktısını ekrana gönderdi sistem.

```
In [69]: 1 x.std()
```

```
Out[69]: 1.8639896349099911
```

```
In [70]: 1 bins, counts = histogram(boot)
```

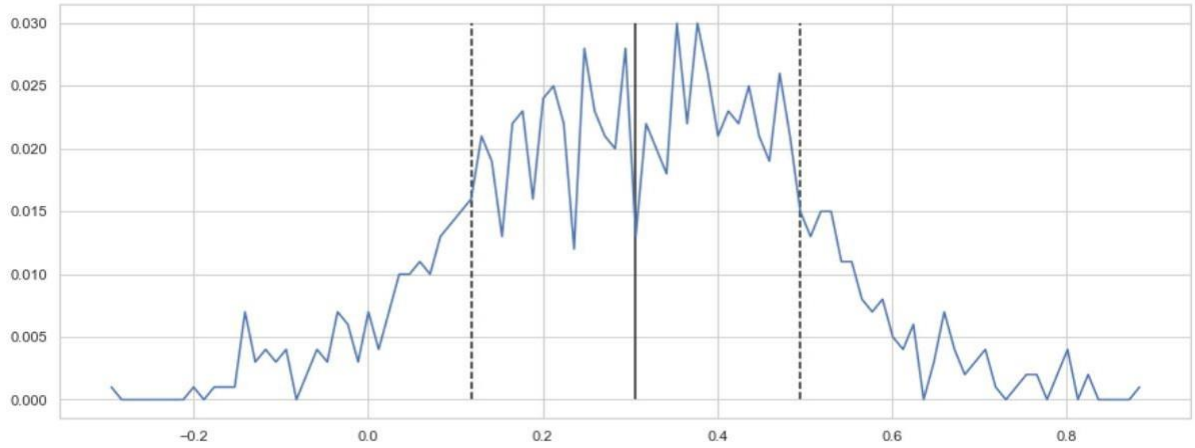
- x.std() yazıp x'in standart sapmasını ekrana yansıtıyoruz.
- boot'un histogramını bins ve counts'a eşitliyoruz.

```
In [71]: 1 plt.plot(bins, counts)
2 plt.vlines(x=boot.mean(), ymin=0, ymax=counts.max(), label='mean')
3 plt.vlines(x=boot.mean()+boot.std(), ymin=0, ymax=counts.max(), label='std', linestyle='--')
4 plt.vlines(x=boot.mean()-boot.std(), ymin=0, ymax=counts.max(), label='std', linestyle='--')
```

- plt ile pyplot kütüphanesini kullanacağız.
- Plt.plot() içerisine bins ve counts değerlerimizi yazdık.
- İlk Plt.vlines ile boot.mean yani orta bölüme y eksenini boyunca kalın çizgi çiziyoruz.
- İkinci Plt.vlines'da boot.mean ile boot.std(boot'un standart sapması)'nı topladığımız yere y eksenini boyunca kesik çizgi çiziyoruz.
- Üçüncü Plt.vlines'da boot.mean ile boot.std(boot'un standart sapması)'nı çıkardığımız yere y eksenini boyunca kesik çizgi çiziyoruz.

Çıktısı;

Out[71]: <matplotlib.collections.LineCollection at 0x28781d210a0>



In [16]:

```
1 # Modeli oluşturalım ve eğitelim
2 def fit_model(train, val, timesteps, hl, lr, batch, epochs):
3     X_train = []
4     Y_train = []
5     X_val = []
6     Y_val = []
7
8     # Eğitim verileri için döngü
9     for i in range(timesteps, train.shape[0]):
10         X_train.append(train[i-timesteps:i])
11         Y_train.append(train[i][0])
12     X_train, Y_train = np.array(X_train), np.array(Y_train)
13
14     # Val verileri için döngü
15     for i in range(timesteps, val.shape[0]):
16         X_val.append(val[i-timesteps:i])
17         Y_val.append(val[i][0])
18     X_val, Y_val = np.array(X_val), np.array(Y_val)
19
20     # Modele Katmanlar Ekleme
21     model = Sequential()
22     model.add(LSTM(X_train.shape[2], input_shape = (X_train.shape[1], X_train.shape[2]), return_sequences = True,
23                   activation = 'relu'))
24     for i in range(len(hl)-1):
25         model.add(LSTM(hl[i], activation = 'relu', return_sequences = True))
26     model.add(LSTM(hl[-1], activation = 'relu'))
27     model.add(Dense(1))
28     model.compile(optimizer = optimizers.Adam(lr = lr), loss = 'mean_squared_error')
29     #print(model.summary())
30
31     # Verilerin eğitimi
32     history = model.fit(X_train, Y_train, epochs = epochs, batch_size = batch, validation_data = (X_val, Y_val), verbose = 0,
33                       shuffle = False, callbacks=callbacks_list)
34     model.reset_states()
35     return model, history.history['loss'], history.history['val_loss']
36
```

- Öncelikle X_train, Y_train, X_val ve Y_val oluşturuyoruz.
- Eğitim verilerimiz için döngü oluşturuyoruz ve burda X_train ile Y_traini kullanıp Numpy Array'a dönüştürüp tekrar kendisine eşitliyoruz.
- Eğitim verilerimiz için döngü oluşturuyoruz ve burda X_val ile Y_val kullanıp Numpy Array'a dönüştürüp tekrar kendisine eşitliyoruz.

- Modele katman ekleme işlemi gerçekleştiriyoruz.
- Bu modelde katman olarak LSTM modeli kullanacağımız için LSTM modelinin katmanlarını yazıyoruz yazıyoruz.
- 4 katmanlı bir LSTM 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 [17]: 1 # Modeli değerlendirme
2 def evaluate_model(model,test,timesteps):
3     X_test = []
4     Y_test = []
5
6     # Verileri test etmek için döngü
7     for i in range(timesteps,test.shape[0]):
8         X_test.append(test[i-timesteps:i])
9         Y_test.append(test[i][0])
10    X_test,Y_test = np.array(X_test),np.array(Y_test)
11    #print(X_test.shape,Y_test.shape)
12
13    # Tahmin
14    Y_hat = model.predict(X_test)
15    mse = mean_squared_error(Y_test,Y_hat)
16    rmse = sqrt(mse)
17    r = r2_score(Y_test,Y_hat)
18    return mse, rmse, r, 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 içerisinde X_test şeklinde yazıyoruz ve Y_hat 'a ekliyoruz.
- Y_test ve Y_hat arasında mse değeri hesaplandı.
- Sqrt ile mse değerinin karekökü alındı ve rmse değeri bulundu.

- R^2_{score} yani belirleme katsayısı, bağımlı değişken içindeki bağımsız değişkenden tahmin edilebilir varyansın oranını hesaplar yani Y_{test} ile Y_{hat} arasındaki varyansın oranı hesaplandı ve r^2 'ye eşitlendi.

```
In [18]: 1 # Tahminleri çizmek
2 def plot_data(Y_test,Y_hat):
3     plt.plot(Y_test,c = 'red')
4     plt.plot(Y_hat,c = 'green')
5     plt.xlabel('Gün')
6     plt.ylabel('Enerji')
7     plt.title('LSTM Tahmin model')
8     plt.legend(['Gerçek','Tahmin'],loc = 'lower right')
9     plt.show()
```

- Plot_error ile güven aralıkları(Y_{test} ve Y_{hat}) ile çizgi çizeceğiz.
- Plt.plot ile Y_{test} 'i yazdıracağız ve rengini kısaltma red ile kırmızı yapıyoruz.
- Plt.plot ile Y_{hat} 'ı yazdıracağız ve rengini kısaltma green ile yeşil yapıyoruz.
- X ekseninde bir etiket oluşturuyoruz ve adını 'Gün' koyuyoruz.
- Y ekseninde bir etiket oluşturuyoruz ve adını 'Enerji' koyuyoruz.
- Grafiğe başlık olarak title ile 'LSTM Tahmin model' yazıyoruz.
- Plt.legend ile çalıştırıyoruz.
- Plt.show ile ekrana gösteriyoruz.
- Unutmamak gerekirkki bu grafiği ileriki kısımlarda tek tek yazmak yerine sadece plot_data yazıp geçeceğiz ve grafiğimiz oluşacak.

```
In [17]: 1 # Eğitim hatalarının grafiğini çizme
2 def plot_error(train_loss,val_loss):
3     plt.plot(train_loss,c = 'r')
4     plt.plot(val_loss,c = 'b')
5     plt.ylabel('Loss')
6     plt.xlabel('Epochs')
7     plt.title('Loss Plot')
8     plt.legend(['train','val'],loc = 'lower right')
9     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 val_loss'u yazdıracağız ve rengini kısaltma b(blue) ile mavi yapıyoruz.
- Y ekseninde bir etiket oluşturuyoruz ve adını 'Loss' koyuyoruz.
- X ekseninde 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 gerekirkı bu grafiğı ileriki kısımlarda tek tek yazmak yerine sadece plot_error yazıp geçeceğız ve grafiğimiz oluşacak.

```
In [18]: 1 # Seriyi ekrana çıkarma
2 series = df[['Global_active_power', 'Global_reactive_power']] # Özellikleri seçtik.2 Giriş.
3 print(series.shape)
4 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 gösterdtk.

```
In [19]: 1 # Train, Val, Test Bölümü
2 train_start = dt.date(2006,12,16)
3 train_end = dt.date(2009,1,24)
4 train_data = series.loc[train_start:train_end]
5
6 val_start = dt.date(2009,1,25)
7 val_end = dt.date(2010,1,25)
8 val_data = series.loc[val_start:val_end]
9
10 test_start = dt.date(2010,1,26)
11 test_end = dt.date(2010,11,26)
12 test_data = series.loc[test_start:test_end]
13
14 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]: 1 # Normalleştirme
          2 sc = MinMaxScaler()
          3 train = sc.fit_transform(train_data)
          4 val = sc.transform(val_data)
          5 test = sc.transform(test_data)
          6 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 [79]: 1 timesteps = 5
          2 hl = [40,35]
          3 lr = 1e-3
          4 batch_size = 16
          5 num_epochs = 400
```

- 'timesteps' ayarını 5 yapıyoruz.
 - 'hl' ayarını 40,35 yapıyoruz.
 - 'lr' ayarını 1e-3 yapıyoruz.
 - 'Batch_size' ile alt küme boyutunu 16 yapıyoruz.
- 'Num_epochs' ile veri kümemizin yalnızca bir kez sinir ağından ileri ve geri aktarılmasıdır. Biz 400 kez ileri geri aktardık.

Not!

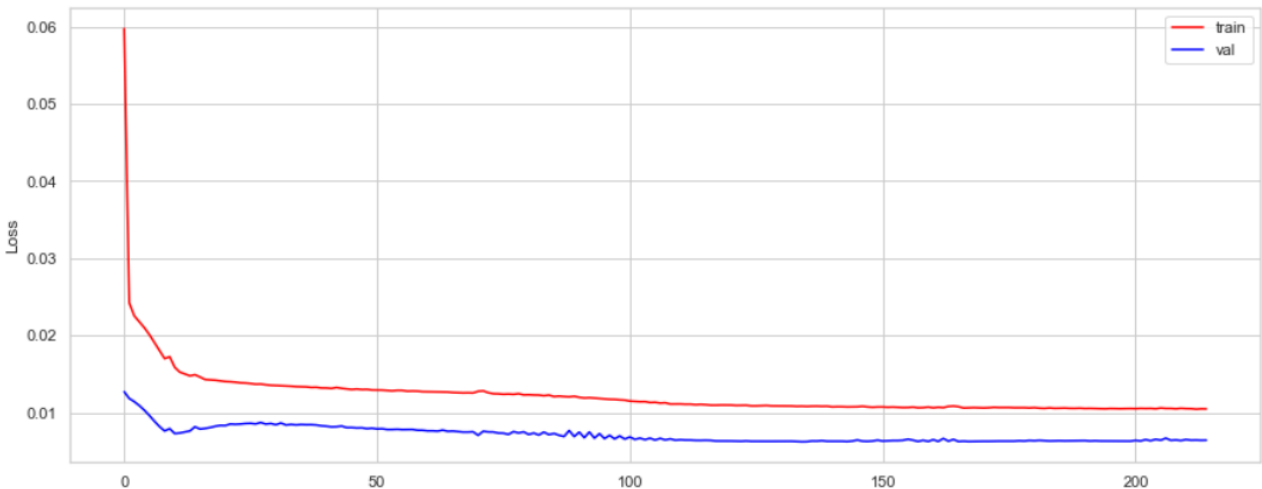
Burada Hiperparametre ayarlarını ayarlıyoruz. Burdaki değişiklik tahmin modelini etkileyecektir.

```
In [80]: 1 #Bu işlem 5-10 dakika arası sürebilir.  
2 model,train_error,val_error = fit_model(train,val,timesteps,hl,lr,batch_size,num_epochs)
```

Epoch 00215: early stopping

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

```
In [81]: 1 plot_error(train_error,val_error)
```



- Burada grafiği yukarıda tanımladığımız kısaltmayı kullanacağız.
- Plot_error diyerek kısaltmamızı kullanıyoruz ve içine çıkan sonuç değerlerini atıyoruz.
- Bu kod bölümünü çalıştırdığımız zaman direk yukarıdaki kod bloğu otomatikmen çalışıp bize sonucu yani çıktıyı verecektir.

```
In [82]: 1 mse, rmse, r2_value,true,predicted = evaluate_model(model,test,timesteps)  
2 print('MSE = {}'.format(mse))  
3 print('RMSE = {}'.format(rmse))  
4 print('R-Squared Score = {}'.format(r2_value))
```

MSE = 0.005906781320399976

RMSE = 0.07685558743773921

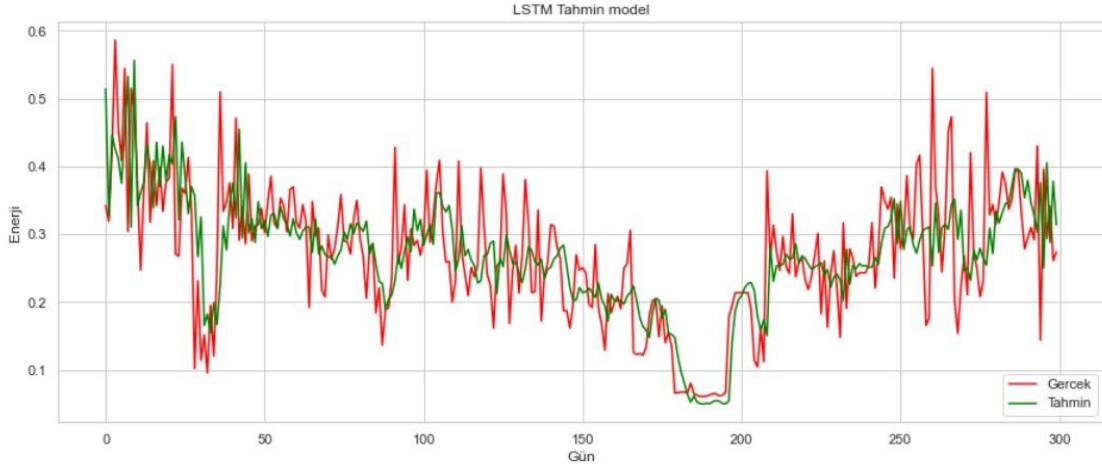
R-Squared Score = 0.41560272321526426

- Mse,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 vs Gerçek Veri

```
In [83]: 1 plot_data(true,predicted)
```



- Buradada grafiği yukarıda tanımladığımız kısaltmayı yani `plot_data`'yı kullanacağız.
- `Plot_data` diyerek kısaltmamızı kullanıyoruz ve içine çıkan tahmin ve gerçek değerlerini atıyoruz.
- Bu kod bölümünü çalıştırdığımız zaman direk yukarıdaki kod bloğu otomatikmen çalışıp bize sonucu yani çıktıyı verecektir.

3.3.5 K-FOLD VE GRIDSEARCHCV

K-FOLD VE GRIDSEARCHCV

```
In [84]: 1 from numpy import mean
2 from numpy import std
3 from sklearn.datasets import make_classification
4 from sklearn.model_selection import KFold
5 from sklearn.model_selection import GridSearchCV
6 from sklearn.ensemble import RandomForestClassifier
7 from sklearn.metrics import accuracy_score
8
```

- Öncelikle önemli kütüphaneleri import ediyoruz.
- Bunlar sklearn kütüphanesi içerisindeki `make_classification`, `KFold`, `GridSearchCV`, `RandomForestClassifier` ve `accuracy_score`.

```
In [85]: 1 #Veri setimizi tekrar tanımlamamızdaki amaç altta x1 ve y1'i ayarlarken bir sorunla karşılaşmamak.
2 df = pd.read_csv('household_daily.csv')
3 df.head()
```

```
Out[85]:
```

	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

- Burada veri setimizi tekrardan tanımlayıp göstermekteki amacımız ileride veri setini ikiye ayıracağımızda sıkıntı olmasın diye.

```
In [86]: 1 x1 = df.iloc[:,1:3]
2 y1 = df.iloc[:,3:4]
```

- Burda veri seti(df) üzerinde iloc ile ayırma işlemi yaptık.
- x1 için satırlar arası 1 ile 3 arasında.
- y2 içinde 3ile4 arasındaki sütunları alacaktır.

```
In [87]: 1 x1
```

```
Out[87]:
```

	Global_active_power	Global_reactive_power
0	1209.176	34.922
1	3390.460	226.006
2	2203.826	161.792
3	1666.194	150.942
4	2225.748	160.998
...
1437	2041.536	142.354
1438	1577.536	137.450
1439	1796.248	132.460
1440	1431.164	116.128
1441	1488.104	120.826

1442 rows × 2 columns

- x1'in görüntüsü yukarıdaki gibidir.

In [88]:

1 y1

Out[88]:

Voltage	
0	93552.53
1	345725.32
2	347373.64
3	348479.01
4	348923.61
...	...
1437	345883.85
1438	346428.76
1439	345644.59
1440	347812.21
1441	303487.57

1442 rows × 1 columns

- y1'in görüntüsü yukarıdaki gibidir.

In [89]:

```
1 import warnings
2 warnings.simplefilter(action='ignore', category=FutureWarning)
3
4 x1_np = x1.to_numpy()
5 y1_np = y1.to_numpy()
6 x1_np, y1_np = make_classification( random_state=1, n_informative=10, n_redundant=10)
7
```

- Öncelikle x1 verimizi ve y1 verimizi numpy array'a dönüştürüp x1_np ve y1_np'ye eşitliyoruz.
- Make_classification ile rastgele bir sınıf oluşturuyoruz
- random_state parametresi train ve test endeksleri içine verilerin bölme karar verecek iç rasgele sayı üreticidir ve biz random_state=1 yapıyoruz.
- bilgilendirici yani informative değerini 10 yapıyoruz.
- Gereksiz yani redundant değerini 10 yapıyoruz.
- Oluşturduğumuz rastgele sınıfı x1_np ve y1_np'ye eşitliyoruz.

```

8 # Çapraz doğrulama prosedürünü yapılandırılalım
9 cv_outer = KFold(n_splits=10, shuffle=True, random_state=1)
10 # Bölmeleri numaralandıralım
11 outer_results = list()
12 for train_ix, test_ix in cv_outer.split(x1_np):
13     # Verileri bölelim
14     X_train, X_test = x1_np[train_ix, :], x1_np[test_ix, :]
15     y_train, y_test = y1_np[train_ix], y1_np[test_ix]
16     # Çapraz doğrulama prosedürünü yapılandırılalım
17     cv_inner = KFold(n_splits=3, shuffle=True, random_state=1)
18     # Modeli tanımlayalım
19     model = RandomForestClassifier(random_state=1)
20     # Arama alanını tanımlayalım
21     space = dict()
22     space['n_estimators'] = [10, 100, 500]
23     space['max_features'] = [2, 4, 6]
24     # Aramayı tanımlayalım
25     search = GridSearchCV(model, space, scoring='accuracy', cv=cv_inner, refit=True)
26     # aramayı yapalım
27     result = search.fit(X_train, y_train)
28     # Tüm eğitim setinde en iyi performans gösteren modeli elde edelim
29     best_model = result.best_estimator_
30     # Geciktirme veri kümesinde modeli değerlendirme
31     yhat = best_model.predict(X_test)
32     # Modeli değerlendirme
33     acc = accuracy_score(y_test, yhat)
34     # Sonucu saklayalım
35     outer_results.append(acc)
36     # İlerlemeleri ekrana yazdıralım
37     print('acc=%.3f, est=%.3f, cfg=%s' % (acc, result.best_score_, result.best_params_))
38 # Modelin tahmini performansını gösterelim
39 print('Accuracy: %.3f (%.3f)' % (mean(outer_results), std(outer_results)))

```

Not: Bir önceki kod bloğuna devam ediyoruz. (Sığmadığı için ayrılmıştır.)

- Öncelikle Çapraz doğrulama yani Kfold'u yapılandırılalım.
- Kfold 'un içerisindeki n_split kısmını 10 yapıyoruz bu bizim katman sayımızı temsil edecektir ve bu bölümü cv_outer'a eşitliyoruz.
- Bölmeleri numaralandırmak için list() ile listeliyoruz ve outer_results'a eşitliyoruz.
- Daha sonra döngü oluşturarak x1_np split'i ile döngü oluşturuyoruz ve içerisine tanımlı train_ix ve test_ix 'i yazıyoruz.
- X_train, X_test, y_train, y_test'i oluşturuyoruz.
- Kfold'u döngü içerisinde tekrar kullanıyoruz ve burada 3 katman kullanıyoruz ve cv_inner'e eşitliyoruz.
- RandomForestClassifier ile model oluşturuyoruz ve model'a eşitliyoruz.
- Daha sonra arama alanını tanımlıyoruz ve space bölümleri oluşturuyoruz.
- İlk olarak space=dict() ile bir sözlük oluşturuyoruz.
- İkinci space ile n_tahmincisini oluşturuyoruz ve tahmin değerlerini 10,100 ev 500 olarak ayarlıyoruz.
- Üçüncü space olarak da maksimum özelliğini oluşturuyoruz ve tahmin değerlerini 2,4,6 ayarlıyoruz.

- Daha sonra Aramayı tanımlayalım ve GridSearchCv ile tanımlayalım burdaki accuracy bizim doğruluk değerimizdir ve bu kısmı search'a eşitliyoruz.
- Search.fit(X_train,y_train) ile aramayı yapıyoruz ve bunuda result'a eşitliyoruz.
- Tüm eğitim setinde en iyi performans gösteren modeli elde etmek için (result.best_estimator_) parametresini kullanıyoruz ve best_model'a eşitliyoruz.
- Geciktirme veri kümesinde modeli değerlendirmek için ise best_model.predict(X_test) parametresini kullanıyoruz ve yhat'a eşitliyoruz.
- Modeli değerlendirmek için accuracy_score() methodunu kullanıyoruz ve içine parametre olarak y_test ve yhat'ı ekliyoruz ve bunuda acc'ye eşitliyoruz.
- outer_results.append()' metodu ile çıkan sonucu saklıyoruz. Yani acc'yi saklıyoruz.
- Print ile acc değerini, en iyi performans verisini(est) ve cfg(tahmin edicileri) ekrana yazdırıyoruz.
- Daha sonra döngüden çıkıyoruz ve en son olarak Modelin tahmini performansını Print ile ekrana yansıtıyoruz.

Çıktısı;

```
acc=0.900, est=0.867, cfg={'max_features': 6, 'n_estimators': 100}
acc=0.900, est=0.867, cfg={'max_features': 2, 'n_estimators': 500}
acc=0.900, est=0.867, cfg={'max_features': 4, 'n_estimators': 500}
acc=0.800, est=0.833, cfg={'max_features': 4, 'n_estimators': 100}
acc=0.900, est=0.867, cfg={'max_features': 6, 'n_estimators': 100}
acc=0.900, est=0.856, cfg={'max_features': 2, 'n_estimators': 100}
acc=0.800, est=0.911, cfg={'max_features': 2, 'n_estimators': 100}
acc=0.800, est=0.856, cfg={'max_features': 2, 'n_estimators': 100}
acc=0.900, est=0.878, cfg={'max_features': 6, 'n_estimators': 100}
acc=0.900, est=0.900, cfg={'max_features': 6, 'n_estimators': 100}
Accuracy: 0.870 (0.046)
```

4 Sonuç

Multivariable(Çok değişkenli) girişli 2006-2011 arasında bir bölgenin kullandığı elektrik tüketimini 'Jupyter Notebook' ile verileri ekrana göstererek daha sonra bu veri seti ile veri görselleştirmesi, güzelleştirmesi ve düzenlemesi yapılmıştır. Öncelikle modelde önce kütüphaneler belirlendi daha sonra veri seti ekrana yansıtılıp veri setini işledikten(düzenledikten) sonra grafikler ile güzelleştirme yapıлып modeller oluşturuldu.. LSTM modelinde gerçek değer ile LSTM 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 model geliştirildi. LSTM modelinde Çeşitli grafiklerle veri görselleştirilmesi yapıldı. Bu modelde MSE ve RMSE değerleri hesaplandı. Karşılaştırma grafiği için matplotlib kütüphanesi kullanıldı. Çeşitli metodlarla verileri hakkında bilgi edinildi. Sonuç olarak bu modelin tahmin grafikleri oluşturuldu. Bu modelde düzenlenen gerçek veriler o tahmin model ile karşılaştırıldı. Daha sonra ekleme yapılarak jackknife ve bootstrap ile veri arttırma teknikleri kullanıldı. Modelin eğitimi tamamlandıktan sonra k katlamalı çapraz doğrulama(k fold cross validation) ile birlikte GridSearchCV ile doğruluk değerlerini ve tahmin edicileri bulduk.

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://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://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://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://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/>
- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [https://ishakdolek.medium.com/lstm-d2c281b92aac#:~:text=LSTM\(Long%2D%20Short%20Term%20Memory, taraf% C4% B1 ndan% 20at% C4% B1 f% 20ald% C4% B1 % 20ve% 20 yayg% C4% B1 nla% C5% 9Ft% C4% B1 r% C4% B1 ld% C4% B1 .](https://ishakdolek.medium.com/lstm-d2c281b92aac#:~:text=LSTM(Long%2D%20Short%20Term%20Memory, taraf% C4% B1 ndan% 20at% C4% B1 f% 20ald% C4% B1 % 20ve% 20 yayg% C4% B1 nla% C5% 9Ft% C4% B1 r% C4% B1 ld% 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-sequence-True-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://en.wikipedia.org/wiki/Gated_recurrent_unit
- 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